VŠB – Technická Universita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Metrika podobnosti formalizací v PL1
The Similarity Measure of PL1 Formalisations

2010 Lukáš Vích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne ……. ……

Touto cestou bych rád poděkoval svému vedoucímu, panu Mgr. Markovi Menšíkovi, Ph.D., za jeho velmi aktivní přístup, okamžité a věcné reakce na mé dotazy a celkovou snahu při jeho vedení. Kromě toho patří mé poděkování také Mgr. Martině Číhalové, která mi během psaní diplomové práce poskytla mnoho cenných rad.

Abstrakt

Tato práce se zabývá problematikou při formalizaci vět z přirozeného jazyka do jazyka predikátové logiky 1. řádu (dále jenom PL1). Cílem je seznámit čtenáře se základy sémantiky PL1, uvést jej do dané problematiky a vysvětlit všechny důležité aspekty spjaté s převáděním vět z přirozeného jazyka do jazyka PL1. Dalším cílem je vymezit konkrétní podmnožinu PL1 a nad ní definovat metriku podobnosti formalizací v PL1 (na jejímž základě bude automaticky vyhodnocena míra správnosti logické formule). Metrika bude definována jako soubor pravidel, která zohledňují ekvivalentní možnosti zápisu (některých) částí logické formule a která navíc ošetřují nejčastější chybování vznikající při formalizování vět do PL1. Konečným cílem práce je implementace algoritmu porovnávající dvě formule (vzor a řešení) a vyhodnocení správnosti ověřované formule (řešení).

Klíčová slova: Predikátová logika 1. Řádu, logická formule, atomická formule, interpretace, formalizace, metrika, gramatika

Abstract

This work deal with questions of sentences formalization from natural language to the language of first order predicate logic (further just PL1). The target is introduce reader to the principles of semantic of PL1, bring him to the existent problems and explain all of the important aspects wedded with sentences conversion from natural langugage to the language of PL1. The next target is define the particular subset of PL1 and define the metric of formalization similarities over it (which will be the base of rate of rightness of logical formula automatically evaluation). The metric will be defined like a set of rules, which are making provison for equivalent entry possibilities (some of these) of logical formula parts and which are making provision for most frequent errancy beyond rising at sentences formalization to the PL1. The final target of this work is implementation of algorithm to compare two formulas (model and solution) and evaluation the similiarity of these formulas.

Keywords: first order predicate logic, logical formula, atomic formula, interpretation, formalization, metric, grammar

Obsah

1. Úvod

V úvodní části bude čtenář seznámen se základními pojmy týkajícími se logického vyplývání (mimo jiné i stručného obsahu celé práce). V druhé kapitole bude věnována pozornost predikátové logice 1. řádu, probere se její sémantika, způsoby formalizace, interpretace formulí, následně se pak objasní některé pojmy týkající se množin, relací a funkcí. Ve třetí kapitole se už bude práce ubírat konkrétním směrem (přiblíží se hlavním cílům této práce), vysvětlí se problematika při automatizovaném přístupu při ověřování logických formulí a představí se vědecká studie pojednávající o nejčastějším výskytu chyb při formalizaci vět z přirozeného jazyka do jazyka predikátové logiky 1. řádu. Ve čtvrté části už bude představen samotný popis vymezené metriky a také popis algoritmu, který na jejím základě bude porovnávat správnost logických formulí. Po této kapitole bude následovat závěr, ve kterém pak budou shrnuty všechny podstatné poznatky této práce.

1.1. Logika na úvod

V případě všech definic celé teoretické části textu vycházím z [1][1]

Logika – co na úvod říct? Ačkoliv je každému čtenáři tento výraz známý, určitě nebude na škodu, když si trochu oživíme podstatu celé věci a přiblížíme srdci čtenáře smysl, proč se vlastně logikou zabývat.

Logika je věda, která se dotýká všech oborů zkoumání. V myšlení, které se odráží v českém jazyce je běžně používaná, aniž by si lidé uvědomovali, že v průběhu života neustále logicky myslí a jednají. Logika je formální věda, zkoumající právě onen způsob vyvozování závěrů – zabývá se zákony logického vyplývání. Logika studuje objektivní podmínky platnosti. Z něčeho, co je pevně stanoveno, vyvozujeme závěry – tedy jedná se o vědu studující relaci „vyplývání“.

Definice 1.1. (logické vyplývání):

Úsudek $P_1, \ldots, P_n / Z$ je deduktivně ***správný (platný)***, značíme $\mathbf{P_1, \ldots, P_n} \models \mathbf{Z}$,jestliže závěr Z ***logicky vyplývá*** z předpokladů $P_1, \ldots, P_n$, tj. za všech okolností takových, že jsou pravdivé všechny předpoklady $P_1, \ldots, P_n$, je (za těchto okolností) pravdivý i závěr Z.

Jako většina jiných věd, logika vznikla na základě filosofie, ovšem razantního uplatnění se dočkala až v matematice a informatice (i když v informatice je to opět spíše záležitost příliš blízká oboru matematiky). Pořád však zůstávají značné rozdíly, tedy stále mějme na paměti, že některé části logiky mají blíže matematice a některé filosofii. My se v této práci seznámíme především s matematickou logikou – konkrétně jedním podoborem – predikátové logiky 1.řádu.

Mějme vždy na paměti, že to, co v logice dokazujeme, dokazujeme vždy pouze na základě určitých předpokladů (premis), které máme k dispozici. Z hlediska matematické logiky zde vůbec nebereme v potaz, zda dané výroky odpovídají skutečnosti. Odvozujeme vždy a pouze z těch tvrzení, která jsou pevně daná. Uvedeme si malý příklad. Potká jednou mrazák žábu a praví: „Když jste pořád žába, tak vás patrně ještě nepolíbil princ.“ Na to žába odpověděla: „Já jsem žába a princ už mě dávno políbil.“ Mrazák se naštvaně otočil a při odchodu jenom zlostně prohodil: „Ale nepolíbil ty ulhaná žábo.“ Zdá se vám to nesmyslné? Ano, jistě, ale o to nám vůbec nejde. My pouze analyzujeme a vyhodnocujeme to, co je pevně dané. Vyberme si z tohoto příběhu několik předpokladů, které následně zformalizujeme. Předpoklady (premisy) z onoho příběhu budou vypadat takto:

[1] Veškeré citace v textu se vztahují k 6. kapitole – seznam použité literatury.

P1: Cokoliv je žába, pak platí, že ji nepolíbil princ (jestliže je žábou, pak jí nepolíbil princ)
P2: Toto individuum je žába a princ ji políbil. (jsem žábou a princ mě políbil)

Z celého příběhu pak intuitivně vyplývá ještě závěr:

Z: Ale nepolíbil ty ulhaná žábo (Žábu nikdo nepolíbil)

Tedy pokud platí předpoklad 1, pak žába v něčem lže. Ve formalizaci úsudku výše však lhaní neuvažujeme, pouze vidíme, že premisy jsou vzájemně sporné. Protože vyplývání je záležitost nutného vztahu mezi premisami, žádnou intuici aplikovat nebudeme.

Formalizací ve výrokové logice bychom dostali na základě těchto výroků tento úsudek:

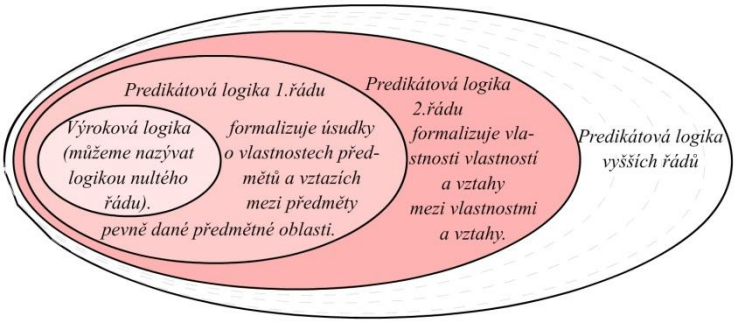
$$\begin{array}{l} P1: Ž \supset \neg P \\ \underline{P2: Ž \wedge P} \\ Z: \quad \neg P \end{array}$$

A tento úsudek je na základě své formy platný, neboť podle definice logického vyplývání platí, že ze sporných předpokladů vyplývá cokoliv, tedy výše uvedený úsudek je platný. Ať se může zdát tento příklad nesmyslný a zbytečný, jedno mu odepřít nemůžeme a to, že na základě výše uvedených premis, je tento úsudek nutně deduktivně správný (platný). Přesně o to nám v logice jde.

2. Predikátová logika 1. řádu

V celé druhé části jsem vycházel ze studijních materiálů podle [2], [3] a [4].

Predikátová logika 1. řádu je zobecněním výrokové logiky. Zavádí a rozšiřuje výrokovou logiku o některé elementy, díky nimž je možné lépe strukturovaně a obecněji vyjádřit určité tvrzení. Ve výrokové logice bylo možné pomocí jistého slovního zápisu vyvodit závěr z množiny daných předpokladů a to vše přepisem jednotlivých vět na matematický zápis. Elementární části (výrokové proměnné) každé věty se navíc mohly (ale nemusely, jelikož formule může obsahovat třeba i jen jednu výrokovou proměnnou – tzv. atomická formule) spojit pomocí logických spojek do tzv. formulí. Tím se zajišťoval formální přepis na tvrzení, které odpovídá souvětím v přirozeném jazyce. Výroková logika umožňuje analyzovat jazyk pouze do úrovně elementárních výroků, nezkoumá hlouběji jejich strukturu a vnitřní vazby mezi jejich vnitřními komponentami a to je právě zásadní rozdíl. Predikátová logika totiž umožňuje navíc analyzovat elementární výroky do úrovně vlastností individuí výroku a jejich vzájemných vztahů.



Obrázek 1: Logika n-tého řádu

Podívejme se na následující příklad, zohledňující některé rozdíly vyjádření ve výrokové logice oproti predikátové.

Všichni velmi bohatí politici jsou čestní. Jiří je velmi bohatý politik. Tedy Jiří je čestný.

Formalizace ve výrokové logice	Formalizace v predikátové logice 1.řádu
Úsudek: **1.** P (Předpoklad 1) **2.** Q (Předpoklad 2) **Z:** R (Závěr)	**Úsudek:** **1.** $\forall x\ [\ (P(x) \wedge B(x)) \supset C(x)\ ]$ **2.** $P(J) \wedge B(J)$ **Z:** C(J)

Tabulka 1: Rozdíly v zápisu výrokové logiky a predikátové logiky 1.řádu

Na první pohled vidíme značné rozdíly mezi oběmi reprezentacemi. Jak je patrné, formalizace z pohledu predikátové logiky je složitější a strukturovanější (obecnější) oproti pohledu výrokové logiky. V daném tvrzení existují jisté vnitřní vazby (vztahy mezi jednotlivými elementy). Výroková logika však nepokrývá tuto skutečnost, zaměřuje se pouze na spojování elementárních částí výroků (přičemž výrok je tvrzení, které má pravdivostní hodnotu) a vyvozování závěru z dané množiny předpokladů, aniž by hlouběji analyzovala interní vztahy částí premis.

Tedy, jak jste jistě usoudili, tento zjevně deduktivně správný (platný) úsudek však formalizací ve výrokové logice platný není. $(P \wedge Q) \models R$.

P	Q	R	$(P \wedge Q)$	$(P \wedge Q) \supset R$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Tabulka 2: Tabulka pravdivostního ohodnocení

Z definice logického vyplývání usoudíme, že úsudek skutečně platný není. Za žádných okolností (myšleno valuaci – tedy přiřazením pravdivostní hodnoty výrokové proměnné) se totiž nemůže stát, že by byly všechny předpoklady pravdivé a závěr nepravdivý, což zde znázorňuje tabulka výše (konkrétně červeně zvýrazněný řádek).

Na základě formalizace v predikátové logice 1. řádu je však tento úsudek opravdu platný. Později si to dokážeme pomocí obecné rezoluční metody (jedná se o jednu z nejjednodušších metod při dokazování platnosti úsudku pro všechny modely v PL1). Její použití však vyžaduje seznámení s problémem, jak tato metoda funguje.

Rezoluční metoda v PL1 je zobecněním základní rezoluční metody výrokové logiky. Je to jedna z procedur (algoritmů), které parciálně rozhodují o tom, zda je daná formule v PL1 tautologie (ovšem nepřímým způsobem – tedy nejdříve znegujeme závěr a poté ověřujeme nesplnitelnost). Pro vybranou formuli je spuštěním algoritmu po konečně mnoha krocích nalezen spor – původní formule je tedy tautologie. V opačném případě, kdy nedojdeme ke sporu (tedy vybraná formule splnitelná je) může algoritmus pokračovat donekonečna. Jestliže pak chceme rozhodnout o tom, zdali je formule A logicky pravdivá, rezoluční metodu použijeme na negaci této formule ($\neg A$) a zjišťujeme, jestli je nesplnitelná. Je-li tomu opravdu tak, algoritmus to zjistí a vydá kladnou odpověď. V opačném případě nemusí algoritmus nikdy skončit. V případě, že chceme zjistit $\{A_1,\ldots,A_n\} \models B$, použijeme rezoluční metodu na formuli ve tvaru $A_1 \wedge \ldots \wedge A_n \wedge \neg B$, protože pokud bude tato formule nesplnitelná, potom bude původní formule $(A_1 \wedge \ldots \wedge A_n) \supset B$ tautologií a vztah vyplývání platí.

Při použití rezoluční metody je ten problém, že musí být formule ve speciálním tvaru – tzv. klausulární (Skolemově) formě. Oproti výrokové logice je převedení na klausulární tvar formule složitější a to tím, že je nutné převést formuli na *prenexní tvar Skolemovy formy* a odstranit z ní všechny kvantifikátory. Složitější je i tvar rezolučního odvozovacího pravidla. Jeho použití vyžaduje simultánní úpravu literálů, tzv. unifikaci.

Definice 1.2.:

Formule A predikátové logiky je v ***prenexním tvaru***, má-li podobu

$$Q_1 x_1 \, Q_2 x_2 \ldots Q_n x_n \, B,$$

kde

- $n \geq 0$ a pro každé $i = 1, 2,\ldots,n$ je Q_i buď všeobecný kvantifikátor $\forall$ nebo existenční $\exists$,
- $x_1, x_2,\ldots,x_n$ jsou navzájem různé individuové proměnné,
- B je formule utvořená z elementárních formulí pouze užitím výrokových funktorů $\neg$, $\wedge$, $\vee$.

Výraz $Q_1x_1\ Q_2x_2...Q_nx_n$ se nazývá *prefix (charakteristika)* a B *otevřeným jádrem (maticí)* formule A v prenexním tvaru.

Definice 1.3.:

Skolemova forma uzavřené formule je prenexní tvar této formule, která neobsahuje žádné existenční kvantifikátory. Skolemova forma vznikne z prenexní formy opakovaným použitím následujících dvou operací (***skolemizací***):

1. $\forall x_1\ \forall x_2...\forall x_n\ \exists y\ A(x_1, x_2,...,x_n, y) \rightarrow \forall x_1\ \forall x_2...\forall x_n\ A(x_1, x_2,...,x_n, f(x_1, x_2,..., x_n))$,
   kde f je nový (v jazyce dosud nepoužitý) n-ární funkční symbol, tzv. ***Skolemova funkce***,
2. $\exists x\ \forall y_1\ \forall y_2...\forall y_n\ A(x, y_1, y_2,...,y_n) \rightarrow \forall y_1\ \forall y_2...\forall y_n\ A(c, y_1, y_2,...,y_n)$,
   kde c je nová (v jazyce dosud nepoužitá) individuová konstanta, tzv. ***Skolemova konstanta***.

Každému eliminovanému existenčnímu kvantifikátoru odpovídá jiná Skolemova funkce nebo konstanta.

Skolemovu formu formule A označíme zápisem A^S.

Literál je atomická formule nebo negace atomické formule (např. $p(f(x))$, $\neg q(y)$).

Klausule je disjunkce literálů (např. $[p(f(x)) \vee \neg q(y)]$).

Konjunktivní normální tvar formule predikátové logiky je prenexní tvar formule, jejíž matice je konjunkce disjunkcí literálů (tj. konjunkce klauzulí).

Disjunktivní normální tvar formule predikátové logiky je prenexní tvar formule, jejíž matice je disjunkce konjunkcí literálů.

Klauzulární forma formule je Skolemova forma, jejíž matice je v klauzulárním tvaru, tj. je konjunkcí klausulí.

Nyní se vraťme k předchozímu příkladu a dokažme platnost úsudku za pomocí rezoluční metody:

1. $\forall x\ [\ (P(x) \wedge B(x)) \supset C(x)\ ]$
2. $P(J) \wedge B(J)$
Z: $C(J)$

Odstraníme implikaci, konjunkty rozdělíme do samostatných řádků a závěr znegujeme. Jestliže poté spojujeme jednotlivé řádky dohromady (i za pomocí korektní substituce) a dosáhneme v konečném počtu kroků prázdné formule, pak jsme ve sporu se závěrem a úsudek bude platný.

1. $(\neg P(x) \vee \neg B(x)) \vee C(x)$
2. $P(J)$
3. $B(J)$
4. $\neg C(J)$
5. $\neg P(J) \vee \neg B(J)$ 1.,4., x/J
6. $\neg B(J)$ 2.,5.
7. spor □ 3.,6.

Úsudek je tedy platný

V následující tabulce si představíme jazyk predikátové logiky. Naleznete zde vše, co vás může u výrazů formalizovaných v PL1 při práci potkat, včetně několika doplňujících poznámek hned pod touto tabulkou. Pro lepší přehlednost si zavedeme následující značení.

Nechť:

F je formule v PL1;
• je logická spojka (¬, ∨, ∧, ⊃, ≡);
δ je kvantifikátor (∀, ∃);
τ je term,

pak:

Gramatika predikátové logiky– udává, jak správně převést (formulovat) tvrzení z přirozeného jazyka do jazyka predikátové logiky 1.řádu.		
1.	**Formule (*F*)**	Atomická ***F*** ***F*** • ***F*** ¬ ***F*** (***F***) **δ** proměnná, … ***F***
2.	**Atomická formule**	Predikát ($\tau_1, \tau_2, \ldots, \tau_n$) $\tau_1 = \tau_2$
3.	**Term (τ)**	Funkce ($\tau_1, \tau_2, \ldots, \tau_n$) Konstanta Proměnná
Jazyk predikátové logiky – sestává z dohodnuté skupiny symbolů vystupujících v atomických formulích, formulích a termech.		
4.	**Logická spojka (•)**	¬, ∨, ∧, ⊃, ≡
5.	**Kvantifikátor (δ)**	∀, ∃
6.	**Konstanta**	J, Jiří, …
7.	**Proměnná**	x, y, z, … (všechny individuové proměnné)
8.	**Predikát**	p, q, r, … (jeOtcem, máBarvu, jeVětšíNeboRovno, …)
9.	**Funkce**	f, g, h, … (otec, druhá mocnina, násobek čísla…)

Tabulka 3: Gramatika a jazyk predikátové logiky 1.řádu

Poznámky:

ad.3) je-li arita (množství individuovaných proměnných, které jsou argumenty funkce nebo predikátu) funkce rovná 0, jedná se o nulární funkční symbol, neboli individuovanou konstantu a značí se a, b, c. Není pravou (logickou) konstantou, neboť podléhá (jako každý funkční symbol) interpretaci.

ad. 4 – 7) jedná se o tzv. logické symboly – jsou jedny z atomických částí tvořených formulí

ad. 8 – 9) jedná se o tzv. speciální symboly – určují specifiku jazyka.

Kromě výše uvedených se při práci setkáme i s tzv. pomocnými symboly – těmi jsou myšleny závorky - (,), [,], {, }.

Pro jazyk predikátové logiky 1. řádu je charakteristické to, že jediný přípustný typ proměnných jsou individuové proměnné. Pouze a jen tyto lze vázat kvantifikátory (v logice 2.řádu jsou povoleny i predikátové proměnné).

Definice 1.4.

Výskyt proměnné x ***ve formuli*** A ***je vázaný***, jestliže je součástí nějaké podformule $\forall x B(x)$ nebo $\exists x B(x)$ formule A.

Proměnná x je vázaná ve formuli A, má-li v A vázaný výskyt. Výskyt proměnné x ve formuli A, který není vázaný, nazýváme ***volný***.

Proměnná x je volná ve formuli A, má-li v A volný výskyt.

Formule, v níž každá proměnná má buď všechny výskyty volné nebo všechny výskyty vázané, se nazývá ***formulí s čistými proměnnými***.

Formule se nazývá ***uzavřenou***, neobsahuje-li žádnou volnou proměnnou. Formule, která obsahuje aspoň jednu volnou proměnnou se nazývá ***otevřenou***.

Nechť $x_1, x_2, \ldots, x_n$ jsou všechny volné proměnné formule A. Potom uzavřenou formuli

$$\forall A =_{df} \forall x_1 \forall x_2 \ldots \forall x_n A \quad \text{resp.} \quad \exists A =_{df} \exists x_1 \exists x_2 \ldots \exists x_n A ,$$

nazýváme ***generálním*** resp. ***existenčním uzávěrem formule*** A.

Symbolem $A(x/t)$ označujeme formuli, která vznikne z formule A ***korektní substitucí termu*** t ***za proměnnou*** x. Má-li být substituce korektní musí splňovat následující dvě pravidla:

- Substituovat lze *pouze za volné výskyty* proměnné *x* ve formuli A a při substituci nahrazujeme *všechny volné výskyty* proměnné *x* ve formuli A.
- Žádná individuová proměnná vystupující v termu t se po provedení substituce *x*/t nesmí stát ve formuli A vázanou (v takovém případě je term t za proměnnou *x* ve formuli A ***nesubstituovatelný***).

Symbolem $A(x_1,x_2,\ldots,x_n / t_1,t_2,\ldots,t_n)$ označujeme formuli, která vznikne z formule A korektními substitucemi x_i/t_i pro i = 1, 2,...,n.

Všechny formule tvaru $A(x_1, x_2,\ldots,x_n / t_1, t_2,\ldots,t_n)$ nazýváme ***instancemi formule*** A.

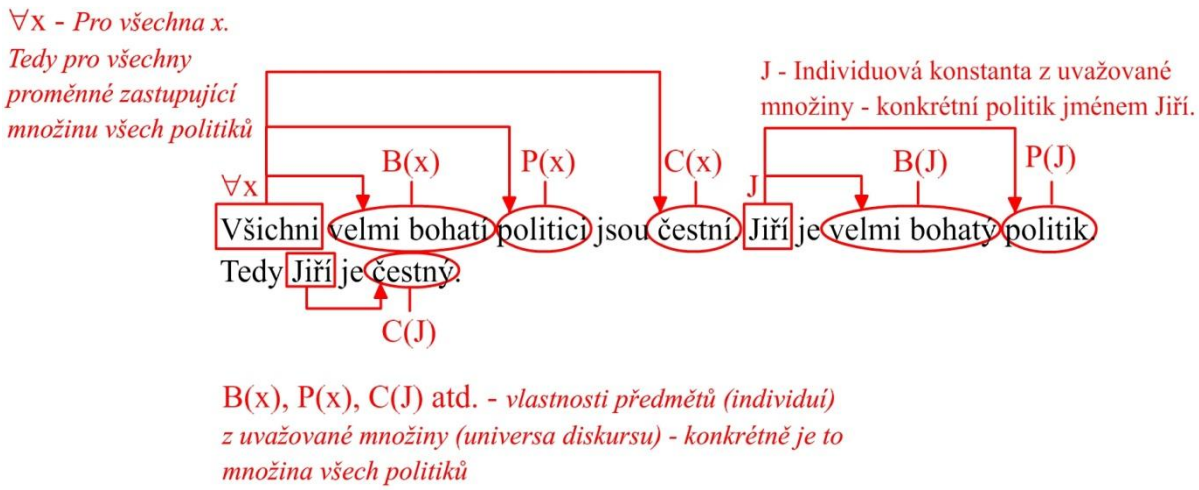
2.1. Převod z přirozeného jazyka do symbolického jazyka PL1

Při převodu nějakých tvrzení z přirozeného jazyka do symbolického jazyka PL1 jde hlavně o správnou analýzu vět. Volba predikátových či funkčních symbolů je libovolná, avšak nesmí dojít ke kolizi vlastností, funkcí, či vztahů. Kvantifikátory využíváme v případě užití výrazů vztahujících se k nějakému počtu médií vystupujících v námi analyzované větě nebo souvětí.

- Všichni, každý, nikdo, … - zastupuje všeobecný kvantifikátor $\forall$.
- Někdo, někteří, existuje, … - zastupuje existenční kvantifikátor $\exists$.

Stále zatím předpokládáme, že jde o jazyk nad homogenním universem, tedy množina všech individuí.

Na následujícím obrázku je popsán způsob intuitivního přepisu slovního zápisu do jazyka predikátové logiky 1.řádu. Vrátíme se již na dříve uvedený příklad s politiky:



Obrázek 2: intuitivního přepisu slovního zápisu do jazyka predikátové logiky 1. řádu

První předpoklad (Všichni velmi bohatí politici jsou čestní) je jakési obecné vyjádření vztahující se na všechny existující politiky. Analýzou této věty bychom ji měli formálně vyjádřit takto:

Pro všechny ($\forall x$) individua (prvků z množiny politiků) platí, že:

> Pakliže je dané individum politik (P(x)) a ($\wedge$) zároveň je bohatý (B(x)), pak ($\supset$) je také čestný (C(x)) - $\forall x$ [(P(x) $\wedge$ B(x)) $\supset$ C(x)]. Konkrétní konstanta z množiny všech politiků – Jiří je politikem (P(J)) a ($\wedge$) zároveň je Jiří velmi bohatý (B(J)). Tedy podle všeho - Jiří je Čestný (C(J)).

Následujících několik příkladů slouží k lepšímu pochopení správné analýzy přirozeného jazyka a následné formalizace do jazyka PL1.

Analyzujte v jazyce PL1 následující výroky:

1) Všichni lidé (C – vlastnost „být člověkem“) na světě jsou buďto dobráci (D – vlastnost „být dobrák“), anebo padouši (P – vlastnost „být padouchem“).

$$\forall x \, [\, C(x) \supset \neg \, (D(x) \equiv C(x)) \,]$$

2) Ne každý člověk (C), co se tváří přívětivě (P), to s vámi myslí opravdu dobře (D).

$$\neg \forall x \, [\, (C(x) \wedge P(x)) \supset D(x) \,]$$

> Ne všichni lidé, co jsou zřejmě lidmi a tváří se přívětivě, pak to s vámi myslí opravdu dobře (uveďme zde ještě, že v každé takto analyzované větě nám záleží na tom, nad jakou množinou (universum) pracujeme. V našem případě se jedná o množinu všech lidí. Podrobněji si však volbu universa a nejen jeho lépe vysvětlíme později – konkrétně u interpretace formulí).

3) Existují však i takoví (bavíme se pořád o lidech), kteří se na první pohled jeví jako padouši, ale ve skutečnosti jsou dobří.

$$\exists x \, [\, (C(x) \wedge P(x)) \wedge D(x) \,]$$

Tedy určitě existuje alespoň někdo, kdo je zřejmě člověkem a jeví se být padouchem a je přitom dobrákem.

Jak je vidět, nezáleží na pojmenování predikátových symbolů, avšak nesmí docházet k interním kolizím ve formuli.

Několik dalších příkladů: Proveďte co nejjemnější formalizaci následujících tvrzení pomocí PL1 (V případě vzniku nejasností ohledně interpretace formule, zvolte příslušné prvky interpretační struktury).

1) Pro všechna přirozená čísla platí, že násobek libovolného čísla s číslem dva je větší nebo roven tomuto číslu. (Za Universum zvolte celá čísla).

$$\forall x\,[P(x) \supset R(f(x,a), x)]$$

Pro všechna přirozená čísla ($\forall x$) platí, že jestliže je číslo přirozené (relace P(x)), pak ($\supset$) násobek libovolného čísla (x) s číslem 2 (a) – tedy jedná se o funkci f(x,a) je větší nebo roven (relace $R(t_1, t_2)$ – t_1 je větší nebo rovno t_2) tomuto číslu (x).

2) Jestliže je násobek dvou přirozených čísel dělitelný dvěma, pak je jedno z nich sudé číslo. (Za Universum zvolte přirozená čísla).

$$\forall x \forall y[D(f(x,y),a) \supset (S(x) \vee S(y))]$$

Pro libovolná 2 čísla x a y platí, že jestliže je násobek těchto dvou čísel (f(x,y)) dělitelný dvěma (a) – tedy relace $D(t_1, t_2)$ – t_1 je dělitelné t_2, pak ($\supset$) je číslo x sudé (S(x)) nebo je sudé číslo y (S(y)).

2.2. Interpretace formulí

Interpretace se zavádí proto, abychom mohli rozhodnout o pravdivosti dané formule, jelikož je význam formule dán právě její interpretací a na jejím základě můžeme rozhodovat o pravdivosti (samozřejmě v definované interpretaci). Tedy sémantika (význam) formulí predikátové logiky je dána právě jejich interpretací. To, jak bude daná hra celá probíhat, jak dopadne, to nám určí naše formule, avšak hráči, jenž v ní budou vystupovat, jejich vlastnosti a pravidla celé hry, na kterých bude hra a výsledek záležet, nám určuje teprve právě definovaná interpretace. Uvedeme krátký příklad:

$$\forall x\ p(f(x), x)$$

Tato formule může v jedné interpretaci být pravdivá a v druhé zase ne. Jestliže si zvolíme universum diskursu přirozená čísla, funkce označená f (…) je druhá mocnina a relace označená p (…) nám říká, že první argument je větší nebo roven druhému. V každé možné valuaci bude tato formule pravdivá, protože pracujeme nad množinou přirozených čísel.
Kdybychom ovšem vytvořili interpretaci jinou a to například tak, že p(…) nyní bude značit, že první argument je ostře menší druhému, pak jasně vidíme, že v této interpretaci bude formule nepravdivá.

Interpretace se tedy skládá ze tří základních bloků:

1) Z definice **Universa diskursu** – jistá neprázdná množina, jejíž prvky jsou individua

2) Z definice **Relace** přiřazená predikátovému symbolu – predikátové symboly vyjadřují vztahy mezi prvky universa. Každému predikátovému symbolu přiřazujeme jistou n-ární relaci (tj. podmnožinu Kartézského součinu) nad universem. Speciálně, jedná-li se o unární predikátový symbol (n = 1), pak přiřazujeme podmnožinu universa.
3) Z definice **Funkce** přiřazené funkčním symbolům – ty opět představují n-ární funkce nad universem.

Teprve poté, co je daná formule interpretována, můžeme vyhodnotit její pravdivost či nepravdivost v dané interpretaci. Pravdivostní hodnota formule nezávisí na hodnotě vázaných proměnných (pouze volné proměnné jsou skutečné proměnné). Obsahuje-li formule nějaké volné proměnné, můžeme vyhodnotit její pravdivost v interpretaci pouze v závislosti na ohodnocení (valuaci) volných proměnných. Při některé valuaci může být formule v dané interpretaci pravdivá, při jiné nepravdivá.

$$\forall x \exists y \; p(f(x), y)$$

Jestliže si zvolíme jako universum množinu celých čísel, relaci p(…) jako 1. argument je větší nebo roven 2. argumentu a funkci f(…) druhou mocninu, pak nám formule říká, že pro každé celé číslo *x* platí, že x^2 je větší nebo rovno *nějakému* číslu *y*. Pravdivost této formule v dané interpretaci pak samozřejmě závisí na ohodnocení proměnné *y*, jelikož ta vůbec nemusí být stejná jako *x*. I kdybychom předpokládali, že je x^2 rovno 1 000 000, kdykoliv může přijít číslo *y*, které je třeba o jedničku větší, tedy relace se neshoduje s jejím původním smyslem a ani formule nebude pravdivá. Pravdivá v daném ohodnocení může být pouze a jen tehdy, jestliže bude číslo *y* záporné anebo rovno 0. V tomto případě rozhodně můžeme říct, že y nikdy nebude větší než druhá mocnina čísla *x*, jelikož druhá mocnina záporného čísla je číslo kladné.

V predikátové logice nám jde vždy o to zjistit, co nám daná formule A říká v závislosti na její interpretaci. Podle struktury formule jsme poté schopni ověřit, zda-li je daná formule A:

Splnitelná v interpretaci I – tedy pokud **existuje** ohodnocení e proměnných takové, že platí $\models_I A\,[e]$.

Pravdivá v interpretaci I (značíme $\models_I A$) – tedy pokud pro **všechna** možná ohodnocení e individuových proměnných platí, že $\models_I A\,[e]$.

Splnitelná – tedy pokud existuje interpretace I, ve které je splněna, tj. jestliže existuje interpretace I a valuace e takové, že $\models_I A[e]$. Takováto interpretace I a valuace e, tedy dvojice <I,e>, pro kterou platí $\models_I A\,[e]$ se nazývá **model** formule.

Tautologií (logicky pravdivá - značíme $\models A$) – tedy pokud je pravdivá v každé interpretaci.

Kontradikcí – tedy pokud neexistuje interpretace I, která by formuli A splňovala (nemá model).

Model množiny formulí $\{A_1,\ldots, An\}$ je taková interpretace I, ve které jsou pravdivé **všechny** formule $A_1,\ldots, A_n$.

Formule B logicky vyplývá z formulí $A_1, \ldots, A_n$, značíme $\mathbf{A_1,\ldots,A_n} \models \mathbf{B}$, jestliže B je pravdivá v každém modelu množiny formulí $A_1,\ldots,A_n$.
Tedy pro každou interpretaci I, ve které jsou pravdivé formule $A_1, \ldots, A_n$ ($\models_I A_1,\ldots, \models_I A_n$) platí, že je v ní pravdivá také formule B ($\models_I B$).

Následující jednoduché formule pomohou objasnit pojem interpretace a jejich vyhodnocení, konkrétně jedná-li se o formuli splnitelnou, logicky pravdivou (tautologii) nebo logicky nepravdivou (nesplnitelnou, kontradikci).

1) $\forall x\ (P(x) \supset R(x,f(x))$

Nejdříve dokažme, že v konkrétní interpretaci je tato formule nepravdivá, tedy zvolme si vhodnou interpretační strukturu:

U = přirozená čísla;
P(x) = relace „být přirozené číslo“;
R(x,f(x)) = relace „být menší“;
f(x) = funkce „druhá mocnina“;

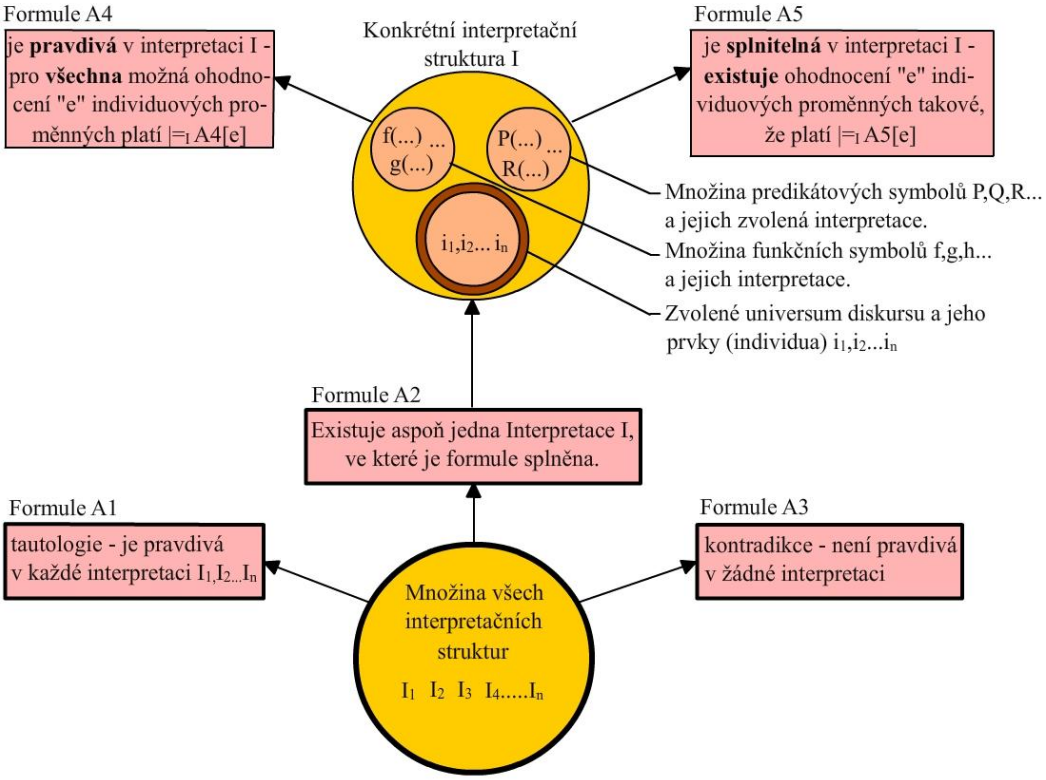
Formule nám tedy říká, že pro všechna přirozená čísla platí, že jsou menší než jejich druhá mocnina. Jelikož je proměnná x vázána na všeobecný kvantifikátor, musí být formule pravdivá pro všechna x, to však není pravda, jelikož např. číslo jedna není menší než jeho druhá mocnina. Kdybychom teď pouze zaměnili relaci R(x,f(x)) za „být menší nebo roven“, pak by tato formule pravdivá byla, avšak i díky této nepatrné změně se jedná o zcela novou interpretaci. Nyní víme, že pro tuto formuli dokážeme nalézt interpretaci I_1, ve které je formule nepravdivá a interpretaci I_2, ve které je formule pravdivá. Tyto podmínky nám stačí k tomu, abychom mohli prohlásit, že tato formule je splnitelná, jelikož existuje interpretace I, ve které je splněna.

2) $\exists x\ (P(x) \vee \neg P(x))$

Uvedená formule je jistě pravdivá v každé interpretaci, jelikož ať už si zvolíme naši interpretační strukturu jakkoliv, formule bude pokaždé pravdivá. Jistě totiž existuje nějaký prvek z universa diskursu, o kterém můžeme prohlásit, že je buďto v P anebo není v P. Jedná se tedy o tautologii. Jedinou podmínkou však je požadavek neprázdnosti universa.

3) $\forall x\ (P(x) \wedge \neg P(x))$

Jinak je to s touto formulí, pro kterou jistě nenalezneme takovou interpretaci, která by ji splňovala. Nikdy nenajdeme interpretaci takovou, kde pro všechny prvky z universa platí, že je x v P a zároveň není v P.



Obrázek 3: Hierarchie interpretační struktury a její vázanost k logickým formul

2.3. Množiny

V následující části textu jsem vycházel ze studijních materiálů podle [5]a [6].

V předchozím výkladu se už o množinách hovořilo, avšak ne podrobně. Následující řádky usnadní pochopení další části této práce, týkající se relací a funkcí.

Množina je obecně chápána jako soubor určitých objektů, který je shrnutý v jeden celek. Přidružené objekty nazýváme prvky dané množiny. Prvky množiny mají obvykle nějakou společnou vlastnost, která je pro ně v rámci množiny charakteristická (např. pro množinu ovocných plodů jsou prvky množiny konkrétním ovocem; pro množinu přirozených čísel je charakteristická nezáporná hodnota prvků množiny; nebo pro prvky množiny iracionálních čísel je charakteristické to, že je není možné přesně vyjádřit s použitím desetinné čárky – mají nekonečný neperiodický vývoj, jako je třeba Ludolfovo číslo apod.). Je-li nějaký objekt *a* prvkem množiny A, zapisujeme tuto skutečnost jako $a \in A$. Není-li objekt *a* prvkem množiny A, zapisujeme $a \notin A$. Z toho vyplývá, že množiny jsou plně určeny svými prvky.

Existují 3 rozdělení množin související s jejich počtem. Jsou to:

a) **Konečná množina** – je taková množina, která obsahuje pouze konečně mnoho různých prvků.
b) **Nekonečná množina** – je taková množina, která obsahuje nekonečný počet různých prvků.
c) **Prázdná množina** – je taková množina, která neobsahuje žádný prvek. Značí se ∅, případně jako dvojice složených závorek {}. Specifickou vlastností prázdné množiny vůči všem ostatním množinám je ta, že prázdná množina je vždy podmnožinou všech ostatních množin.

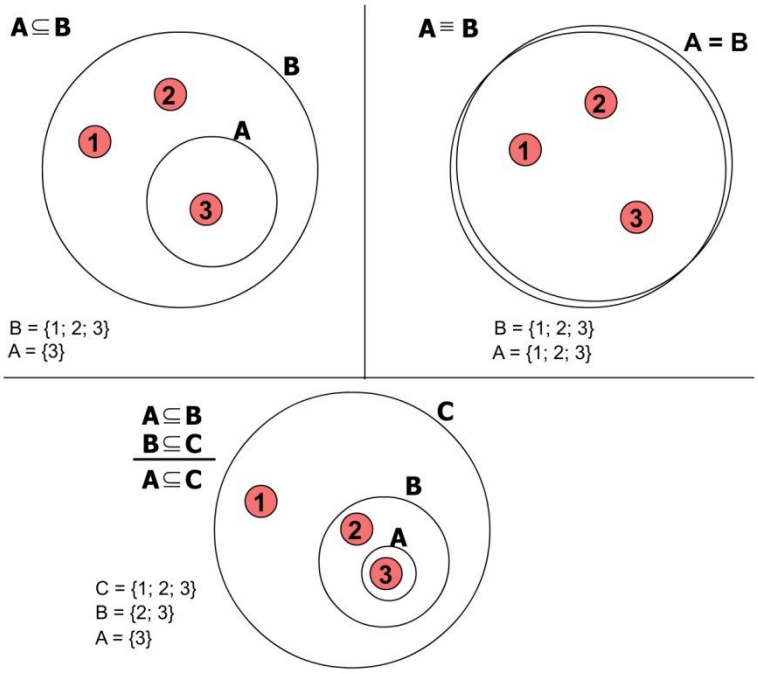
Mezi samotnými množinami existují vztahy:

a) **Inkluze** – řekněme, že množina A je podmnožinou množiny B, jestliže je prvek z množiny A také prvkem množiny B. Značíme $A \subseteq B$. Jinak řečeno, pro každý objekt x platí, že je-li prvkem množiny A, pak je také prvkem množiny B.

$$A \subseteq B <=> \forall x\,[(x \in A) \supset (x \in B)]$$

Z uvedeného pak můžeme odvodit vztah: $A \equiv B <=> [(A \subseteq B) \wedge (B \subseteq A)]$

Z čehož vyplývá, že pro jakékoliv množiny A, B, C platí: $[(A \subseteq B) \wedge (B \subseteq C)] \supset (A \subseteq C)$



Obrázek 4: Inkluze a rovnost množin

Platí-li současně vztah mezi množinami $A \subseteq B$ a zároveň $A \neq B$, zapisuje se vztah zkráceně $A \subset B$.

b) **Rovnost** – řekněme, že dvě množiny A, B považujeme za shodné (sobě rovné), právě když každý prvek množiny A je zároveň prvkem množiny B. Tedy jsou obě množiny tvořeny stejnými prvky. Jinak řečeno, pro každý objekt x platí, že x je prvkem A právě tehdy a jen tehdy, je-li prvkem B.

$$A \equiv B <=> \forall x\,[(x \in A) <=> [(x \in B)]$$

Množiny se zapisují dvěma základními způsoby:

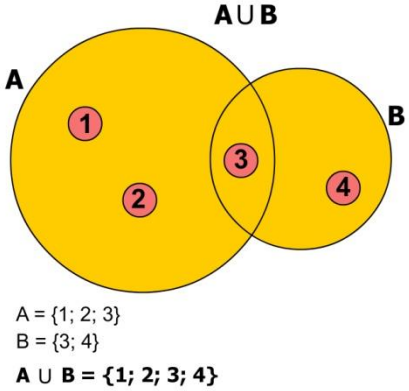
a) **Výčtem prvků**: množina obsahující prvky $a_1, \ldots, a_n$ se označuje jako $\{a_1, \ldots, a_n\}$. Tento zápis můžeme užívat u zapisování konečných množin (např. množina o 3 prvcích {1, 2, 3})
b) **Charakteristickou vlastností**: všechny prvky x spadající do této množiny musí splňovat určitou vlastnost $\varphi(x)$. Obvykle se značí jako $\{x \mid \varphi(x)\}$. Charakteristická vlastnost může být

popsána i v přirozeném jazyce, ale musí být jednoznačně určená (např. všechna čísla *x*, která jsou lichá a jsou větší než 0 a zároveň menší než 10. Z tohoto slovního popisu nám vzejde množina o 5ti prvcích {1, 3, 5, 7, 9}). Charakteristickou vlastností můžeme zapisovat nekonečné množiny (např. {x | x je liché}).

Nad množinami můžeme dále provádět různé operace:

a) **Sjednocení množin A U B**: je definováno jako množina, která je tvořena těmi prvky, které náleží buď množině A anebo množině B. Tedy těmi prvky, které patří jak do množiny A, tak do množiny B.

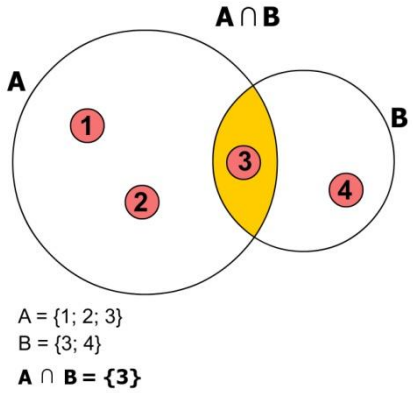
$$A \cup B = \{ x \mid (x \in A) \vee (x \in B) \}$$



Obrázek 5: Sjednocení množin

b) **Průnik množin A ∩ B**: je definován jako množina, která je tvořena těmi prvky, které náleží současně množině A i množině B, tedy obsahuje prvky, které jsou stejné pro obě množiny.

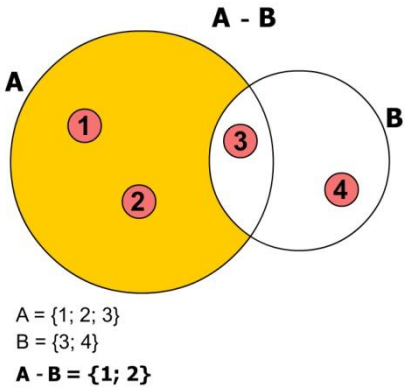
$$A \cap B = \{ x \mid (x \in A) \wedge (x \in B) \}$$



Obrázek 6: Průnik množin

c) **Rozdíl množin A – B**: je definován jako množina, která je tvořena prvky,které náleží množině A a zároveň neleží v množině B, tedy těch prvků množiny A, které nejsou prvky množiny B.

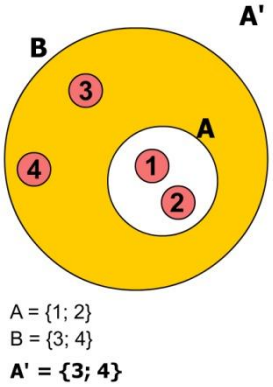
$$A - B = \{ x \mid (x \in A) \wedge (x \notin B)\}$$



Obrázek 7: Rozdíl množin

d) **Doplněk množiny A (v základní množině B) A‘**: je definován jako inverzní množina k množině A, tedy konkrétně doplněk A‘ na množině B je (za předpokladu, že A $\subset$ B, tedy A je nevlastní podmnožinou množiny B) rozdíl množin B – A (např. doplněk přirozených čísel N = {0, 1, 2, …, n} ke množině celých čísel jsou všechna ta čísla, která nepatří do množiny přirozených čísel, ale přitom patří do množiny celých čísel. Tedy se bude jednat o všechna čísla se záporným znaménkem z množiny celých čísel: N‘ = {- n, …, -2, -1}).

$$A' = \{ x \mid (x \notin A) \wedge (x \in B)\}$$

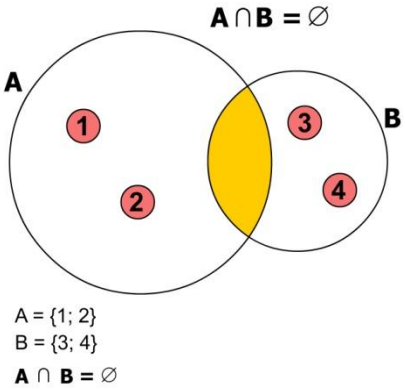


Obrázek 8: Doplněk množiny

Pro libovolnou množinu **A** $\subseteq$ **B** pak například platí tyto vztahy:

(A U A‘) = B,
(A ∩ A‘) = ∅,
A‘‘ = A

e) **Disjunktní množiny A ∩ B = ∅**: jsou takové množiny A a B, jejichž průnik je roven nule, tedy jsou to takové množiny, které nemají žádné společné prvky.



Obrázek 9: Disjunktní množiny

Mezi operacemi na množinách platí také řada rovností. Zejména následující zasluhují pozornost:

a) **Komutativita:**
(A ∪ B) = (B ∪ A)
(A ∩ B) = (B ∩ A)

b) **Asociativita:**
(A ∪ B) ∪ C = A ∪ (B ∪ C)
(A ∩ B) ∩ C = A ∩ (B ∩ C)

c) **Idempotence:**
A ∪ A = A
A ∩ A = A

d) **Distributivita:**
A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C)
A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)

e) **De-Morganovy pravidla**
1. A - (B ∩ C) = (A - B) ∪ (A - C)

Důkaz:
Nechť $x \in (A - (B \cap C)) \Rightarrow (x \in A) \wedge x \notin (B \cap C)$
Pak $x \in A \wedge ((x \notin B) \vee (x \notin C)) \Rightarrow ((x \in A) \wedge (x \notin B)) \vee ((x \in A) \wedge (x \notin C)$
Tedy $(x \in (A - B)) \vee (x \in (A - C)) \Rightarrow$ **$x \in$ (A - B) U (A - C)**

2. A - (B ∪ C) = (A - B) ∩ (A - C)

Důkaz:
Nechť $x \in (A - (B \cup C)) \Rightarrow (x \in A) \vee x \notin (B \cup C)$
Pak $x \in A \vee ((x \notin B) \wedge (x \notin C)) \Rightarrow ((x \in A) \vee (x \notin B)) \wedge ((x \in A) \vee (x \notin C))$
Tedy $(x \in (A - B)) \wedge (x \in (A - C)) \Rightarrow$ **$x \in$ (A - B) ∩ (A - C)**

2.4. Relace a funkce

Relace je matematickým zápisem pojmu „vztah“ platící pro různé objekty (např. číslo 1 je ve vztahu „být menší než“ s číslem 2). Vztah je určen aritou, tedy množstvím objektů, které do vztahu

vstupují (např. do vztahu „být otcem“ vstupují 2 objekty – jedná se proto o relaci binární). Relací (n-ární) je v matematice označován vztah mezi skupinou prvků jedné nebo více množin. V rozsahu této práce se budu zmiňovat pouze o unární či binární relaci (ternární a vyšší řády nebudou probírány).

Definice 1.5. (relace)

n-ární relací mezi množinami M_1, M_2, M_3,..., A_n, kde n $\in$ **N**, rozumíme libovolnou podmnožinu kartézského součinu $\mathbf{R} \subseteq \mathbf{M_1} \times \mathbf{M_2} \times \mathbf{M_3} \times \mathbf{M_n}$.

Právě podle počtu množin kartézského součinu se relace rozdělují na unární, binární ternární a vyšší.

Základním kamenem při tvorbě kartézských součinů množin a relací mezi množinami je pojem uspořádané n-tice prvků. Uspořádaná proto, že v této n-tici záleží na pořadí prvků.

Definice 1.6. (Kartézský součin)

Kartézský součin A × B je množina všech uspořádaných dvojic <a, b>, kde a $\in$ A, b $\in$ B

Př.: Kartézský součin množin A = {a_1, a_2, a_3}, B = {b_1, b_2, b_3} se rovná:

A × B = {<a_1, b_1>, <a_1, b_2>, <a_1, b_3>, <a_2, b_1>, <a_2, b_2>, <a_2, b_3>, <a_3, b_1>, <a_3, b_2>, <a_3, b_3>}

Př.: Relace je podmnožinou kartézského součinu A × B – např.: R $\subseteq$ A × B = {<a_1, b_1>, <a_1, b_2>, <a_2, b_1>, <a_2, b_3>, <a_3, b_3>}

Následující obrázek ilustruje relaci R:

Obrázek 10: Podmnožina Kartézského součinu množin A a B

Notace: <a, b> $\in$R značíme také prefixně R(a, b) nebo infixně a R b.

2.4.1. Základní vlastnosti binárních relací:

a) Relace R je ***reflexivní***: Každý prvek je v relaci sám se sebou.

$\forall x$ (R(x,x))

(Opak reflexivní relace je relace ***ireflexivní***: $\forall x$ ($\neg$R(x,x))

b) Relace R je ***symetrická***: Je-li první v relaci s druhým, pak druhý je v relaci s prvním.

$\forall x \forall y$ [R(x,y) $\supset$ R(y,x)]

c) Relace R je ***anti-symetrická***: Je-li první v relaci s druhým a druhý je v relaci s prvním, pak první je identický s druhým.

$\forall x \forall y [(R(x,y) \wedge R(y,x)) \supset x=y]$

d) Relace R je ***asymetrická***: Je-li první v relaci s druhým, pak druhý není v relaci s prvním.

$\forall x \forall y [R(x,y) \supset \neg R(y,x)]$

e) Relace R je ***transitivní***: Je-li první v relaci s druhým a druhý v relaci s třetím, pak první je v relaci s třetím.

$\forall x \forall y \forall z [(R(x,y) \wedge R(y,z)) \supset R(x,z)]$

$\forall x \forall y \forall z [R(x,y) \supset (R(y,z) \supset R(x,z))]$

f) Relace R je ***cyklická***: Je-li první v relaci s druhým a druhý se třetím, pak třetí je v relaci s prvním.

$\forall x \forall y \forall z [(R(x,y) \wedge R(y,z)) \supset R(z,x)]$

g) Relace R je (***lineární***) ***souvislá***: Je-li první v relaci se druhým nebo je druhý v relaci s prvním nebo jsou identické.

$\forall x \forall y [R(x,y) \vee R(y,x) \vee x=y]$

2.4.2. Operace s binárními relacemi:

Díky speciální struktuře binárních relací s nimi lze provádět i další operace. Ukažme si např. zápis binární relace R mezi dvěmi množinami A = {1, 2, 3, 4} a B = {a, b, c, d} pomocí tabulky. Relace R mezi těmito množinami je definována jako:

R = {<1, a>,<2, a>, <1, b>, <4, b>, <2, c>, <3, c>, <4,d>}.

Je-li $<x,y> \in R$, je v průsečíku řádku *x* a sloupce *y* symbol ○.

R	a	b	c	d
1	○	○		
2	○		○	
3			○	
4		○		○

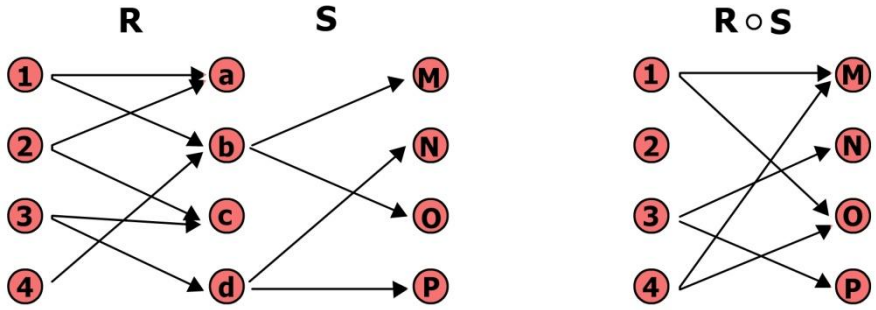
Tabulka 4: Zápis binární relace R mezi dvěma množinami

Inverzní relace R^{-1} k relaci $R \subseteq X \times Y$ je relace mezi množinami $Y \times X$, která je definována předpisem: $R^{-1} = \{<y, x> \mid <x, y> \in R\}$. Například inverzní relace k relaci R z předchozí tabulky je R^{-1} = {<a, 1>,<a, 2>, <b, 1>, <b, 4>, <c, 2>, <c, 3>, <d,4>}.

Skládání relace – je-li R relace mezi množinami X a Y a dále S relací mezi množinami Y a Z, pak složení relací R a S je relace $R \circ S$ mezi množinami X a Z, která je definována jako:

$$R \circ S = \{<x, z> \mid \exists y \in Y: ((<x, y> \in R) \wedge (<y, z> \in S))\}$$

Tedy <x, z> je složená relace R ○ S, jestliže existuje prvek y ∈ Y takový, že <x, y> jsou v relaci R a <y, z> jsou v relaci S. Skládání relací nejvhodněji vyjádříme na následujícím obrázku.



Obrázek 11: Skládání relací

Relace R je definována R = {<1, a>,<2, a>, <1, b>, <4, b>, <2, c>, <3, c>, <4,d>}.
Relace S je definována S = {<b, M>, <d, N>, <b, O>, <d, P>}.
Složená relace R ○ S má pak tvar R ○ S = {<1, M>, <4, M>, <3, N>, <1, O>, <4, O>, <3, P>}.

Definice 1.7. (funkce)

Nechť jsou A a B dvě množiny. *Funkce* nebo *zobrazení f z A do B* je relace z A do B, pro kterou je každý prvek x ∈ A v relaci přesně s jedním prvkem y ∈ B. Tedy:

$$<x, y> \in R$$

A pro každé x ∈ A a y_1, y_2 ∈ B platí:

$$[(<x, y_1> \in R) \wedge (<x, y_2> \in R)] \supset y_1 = y_2$$

Funkce je matematickým vyjádřením pro pojem přiřazení. Objektům jsou často jednoznačným způsobem přiřazeny jiné objekty (např. funkce sinus přiřazuje každému číslu *x* objekt sin(*x*)). Tedy přiřazení vyjadřuje množinu dvojic <*x*,*y*>, kde *y* je objekt přiřazený objektu *x*. Je tedy možné si přiřazení představit jako binární relaci mezi množinou objektů A, kterým jsou přiřazovány jiné objekty a množinou objektů B, které jsou objektům z A přiřazovány. Funkce je tedy speciálním případem relace, která má díky jednoznačnosti přiřazení speciální vlastnost: jednoznačnost přiřazeného objektu (tzn. objektu *x* nemohou být přiřazeny dva rozdílné objekty).

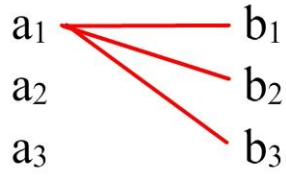
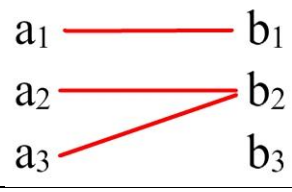
Funkce jsou relace mezi A a B, kde každý prvek z A je v relaci právě s jedním prvkem z B. Využijeme-li termíny pro relace, můžeme ekvivalentně definovat:

Funkce je relace mezi A a B, kde každý prvek z A je v relaci právě s jedním prvkem z B.

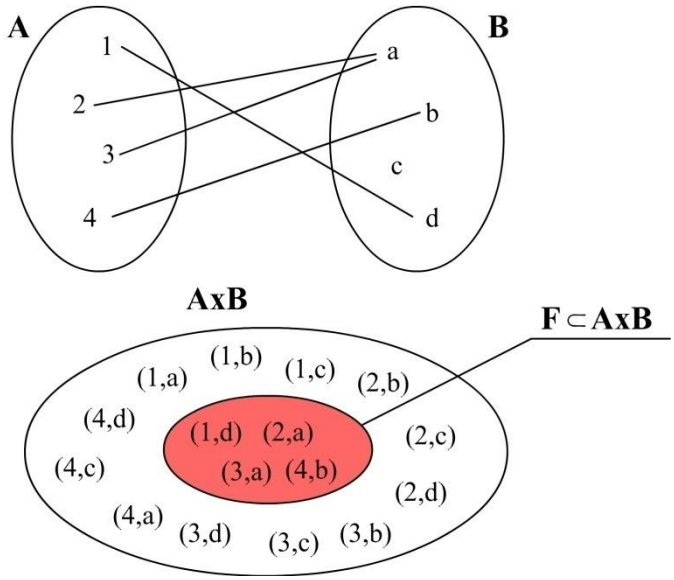
Obdobně lze také definovat parciální funkce následujícím způsobem:

Parciální funkce je relace mezi A a B, kde každý prvek z A je v relaci nejvýše s jedním prvkem z B.

Jak už bylo řečeno, funkce je speciálním případem relace. Jedná se o zobrazení F z množiny A do množiny B (F: A → B), přičemž je zobrazení zprava jednoznačné (každému prvku z $a \in$ A je přiřazen nejvýše jeden prvek $b \in$ B). Na obrázku níže to lépe objasníme:

Vzor (def.obor) Obraz (obor hodnot)	Popis:
	Nejedná se o funkci (resp. zobrazení), protože není zprava jednoznačná
	Je funkcí (resp. zobrazením), je zprava jednoznačná $F(a_1)=b_1$, $F(a_2)=b_2$, $F(a_3)=b_2$

Tabulka 5: Zobrazení zprava nejednoznačné (resp. jednoznačné)



Obrázek 12: Funkce jako speciální případ relace

V PL1 používáme jako interpretaci funkčních symbolů formulí pouze *totální* funkce: ke každému prvku a ∈ A existuje *právě jeden* prvek b ∈ B.

$$\forall a\, \exists b\, [F(a) = b] \wedge \forall a \forall b\, \forall c\ [((f(a) = b) \wedge (f(a) = c)) \supset (b = c)]$$

Uveďme příklad, na kterém demonstrujeme rozdíl mezi Relaci a funkcí:

Př.: Mějme dvě množiny X = {1, 2, 3} a Y = {a, b, c, d}. K daným množinám vytvoříme 3 různé relace a rozhodneme, jestli se jedná jejich speciální případ - o funkce.

Relace R = {<1, a>; <2,b>} – není funkce X → Y, protože k prvku z množiny X (konkrétně 3 ∈ X) neexistuje prvek ležící v množině Y tak, že <x, y> ∈ R.

Relace R = {<1,a>; <2,c>, <3, b>, <3, d>} – není funkce X → Y, protože k prvku z množiny X (konkrétně 3 ∈ X) existují 2 různé prvky, které jsou s ním v relaci (prvky b, d ∈ Y). Očividně je jasné, že b ≠ d.

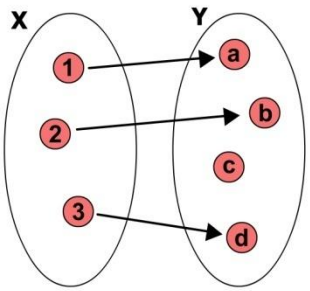
Relace R = {<1, a>; <2, b>, <3, d>} – je funkce X → Y, protože jednoznačně přiřazuje každému prvku z množiny X právě jeden prvek z množiny Y.

2.4.3. Surjekce, injekce, bijekce

Pojmy surjekce, injekce a bijekce jsou konkrétní typy zobrazení (resp. funkcí) a vyjadřují specifické vlastnosti zobrazení.

Injekce (prosté zobrazení množiny X do množiny Y) je takové, pro které každé x_1, $x_2 \in X$ platí, že když $x_1 \neq x_2$ plyne $f(x_1) \neq f(x_2)$.

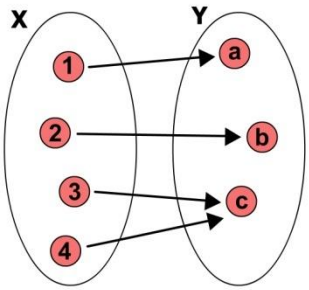
$$\forall x_1 \forall x_2 [(X(x_1) \wedge X(x_2) \wedge (x_1 \neq x_2)) \supset (f(x_1) \neq f(x_2)]$$



Obrázek 13: Injekce

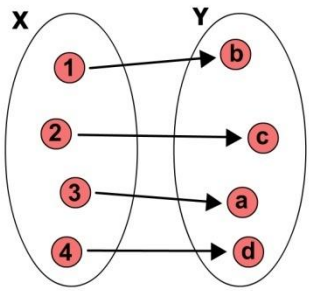
Surjekce (zobrazení množiny X na množinu Y) je takové, pro které platí, že právě pro všechny $y \in X$ existuje $x \in X$ takové, že $f(x) = y$

$$\forall y \, [(Y(y) \supset \exists x \, (X(x) \wedge (f(x) = y)]$$



Obrázek 14: Surjekce

Bijekce (vzájemně jednoznačné nebo také prosté zobrazení X na Y) jestliže je *f* surjekce a injekce zároveň. Existuje-li mezi množinami X, Y bijekce, potom mají stejný počet prvků.



Obrázek 15: Bijekce

Pro lepší pochopení poslouží následující příklad.

Př: Mějme množiny X = {1, 2, 3, 4} a Y = {a, b, c, d}. Definujeme si nad těmito množinami funkci f = {<1, a>; <2, a>; <3, d>; <4, c>}.

Funkce f není injektivní, protože sice $f(1) = f(2)$, ale $1 \neq 2$.
Funkce f není surjektivní, protože neexistuje $x \in X$ takové, že $f(x) = b$.
Funkce f není bijektivní, protože f není surjekce a injekce.

Př: Mějme množiny X = {1, 2, 3, 4} a Y = {a, b, c, d}. Definujeme si nad těmito množinami funkci f = {<1, b>; <2, a>; <3, d>; <4, c>}.

Funkce f je zřejmě injektivní (podle vztahu že pro které každé $x_1, x_2 \in X$ platí, že když $x_1 \neq x_2$ plyne $f(x_1) \neq f(x_2)$).
Funkce f je surjektivní, protože ke všem prvkům (obrazům) z množiny Y umíme najít jeho vzor z množiny X.
Funkce f je bijektivní, protože je injektivní a surjektivní zároveň

3. Automatizovaný přístup porovnávání převodu vět přirozeného jazyka do jazyka logiky

Jak už název nadpisu napovídá, v této části se budu snažit přiblížit způsob, jak by bylo možné automaticky vhodně vyhodnocovat, zda je vypracované studentovo řešení formalizace přirozeného jazyka do jazyka logiky správné (což je v podstatě základním cílem této diplomové práce). Na jednu stranu by se mohlo zdát, že úplně postačí pouze zpracovat algoritmus pro porovnání dvou stringů (řetězců) a na základě jejich podobnosti vyhodnotit jejich správnost. Takové řešení je samozřejmě značně nedostatečné. Hlavním problémem je, že v jazyce logiky nám nejde o slohovou ucelenost nebo gramatické chyby, ale o logickou strukturu analyzované věty. Je také dobře známo, že jak v přirozeném jazyce, tak v jazyce logiky je možné větu napsat více způsoby (tomu se říká parafrázování). U věty přirozeného jazyka řekneme, že jedna věta je parafrází[2] druhé. V matematické logice řekneme, že věty jsou navzájem ekvivalentní (ačkoliv stanovit, jak se co nazývá je spíše oborem lingvistiky než této práce). Proto je nutné brát v potaz to, že existuje více možností, jak formálně zapsat stejnou větu přirozeného jazyka. Pro příklad uveďme následující větu a její možnosti zápisu:

„Všichni lidé, kteří studují vysokou školu, mají maturitu.“ Větu zformalizujeme takovým způsobem, že pro všechna individua (pro všechna „x“ - z Universa všech lidí) platí, že jestliže studují vysokou školu (predikátový symbol P), pak musí mít maturitu (predikátový symbol M).

$$\forall x\,(S(x) \supset M(x))$$

Parafrázováním bychom mohli docílit i jiné formy. Například bychom mohli říct, že: *„Všichni lidé mají maturitu nebo nestudují vysokou školu.“* Zde je například namísto implikace využitá logická spojka „nebo“. Dodejme ještě, že se nejedná o exkluzivní (výlučné) slovní spojení.

$$\forall x\,(M(x) \vee \neg S(x))$$

Parafrázovat se dá různými způsoby (což je značně individuální, jelikož každý člověk uvažuje jinak), obecně však platí základní zákony matematické logiky pro převod implikace (případně ekvivalence) a využití De-Morganových zákonů při formalizaci (pomáhají při negování). V následující tabulce si je ukážeme.

Zákony pro převody	
1.	$\mathbf{A} \supset \mathbf{B} <=> \neg\mathbf{A} \vee \mathbf{B}$
2.	$\neg(\mathbf{A} \supset \mathbf{B}) <=> \mathbf{A} \wedge \neg\mathbf{B}$ **(negace implikace)**
3.	$A \equiv B <=> (\mathbf{A} \supset \mathbf{B}) \wedge (\mathbf{B} \supset \mathbf{A})$
De-Morganovy zákony	
1.	$\neg(\mathbf{A} \wedge \mathbf{B}) <=> \neg\mathbf{A} \vee \neg\mathbf{B}$
2.	$\neg(\mathbf{A} \vee \mathbf{B}) <=> \neg\mathbf{A} \wedge \neg\mathbf{B}$

Tabulka 6: De-Morganovy zákony a zákony pro převody

[2] Parafrázování je někdy vhodné využívat, když si nejsme původně jisti, jak danou větu formalizovat. Hrát si s rozdílnými parafrázemi do doby, než nalezneme ekvivalentní originálu takovou, které lépe rozumíme, nám může pomoci při formalizaci vět z přirozeného jazyka do jazyka logiky.

Zákony pro implikaci	
1.	**A ⊃ B** <=> **¬A ∨ B**
2.	**¬(A ⊃ B)** <=> **A ∧ ¬B (negace implikace)**
3.	A ≡ B <=> **(A ⊃ B) ∧ (B ⊃ A)**

Tabulka 7: Zákony pro implikaci

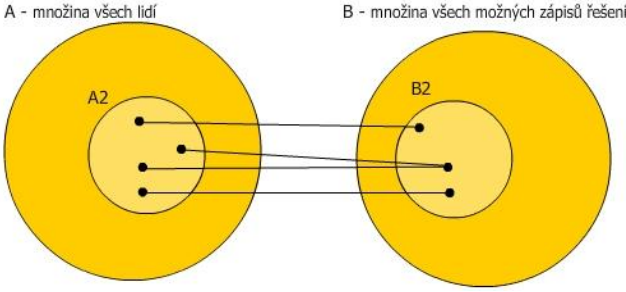
Už jen z uvedeného vyplývá, že formalizace nemůže být jednoznačná a tedy při automatizaci porovnávání správné odpovědi s odpovědí studenta je zapotřebí brát v potaz víc než jenom rize jednu možnost. Generování všech možných ekvivalentních zápisů formule však není náplní této práce. Základní pravidla při tvorbě zadání by měly splňovat tyto parametry:

1. Formule musí být uzavřená (tzn. nesmí se v ní vyskytovat volné proměnné, každá proměnná musí být vázána na určitý kvantifikátor).
2. Formule musí být v prenexním tvaru.
3. Formule musí obsahovat maximálně 3 různé proměnné (spjaté s kvantifikátory).
4. Formule musí obsahovat maximálně 3 různé predikátové symboly.
5. Formule musí obsahovat maximálně 3 různé funkční symboly.
6. Formule musí obsahovat maximálně binární predikátové symboly (jejich arita je větší než 0 a menší nebo rovna 2).
7. Formule musí obsahovat maximálně binární funkční symboly (podobně jako u predikátových).

Na základě těchto pravidel bude poté generována dobře utvořená formule, která by měla být vzorem pro studentovo řešení, to ovšem pořád nezaručuje jednoznačnost řešení převodu vět z přirozeného jazyka do jazyka logiky, ani když jednoznačně definujeme interpretační strukturu.

3.1. Analýza problémových částí při formalizaci vět přirozeného jazyka do jazyka logiky

Snažit se zachytit všechna možná (lidská) řešení, které mohou nastat při formalizaci vět přirozeného jazyka do jazyka matematické logiky, je skoro tak podobné, jako se snažit vyjádřit iracionální číslo konečným počtem číslic. V našem případě však můžeme předpokládat, že se musíme snažit analyzovat ta řešení, která vypracují pouze takoví lidé, kteří jsou se základní problematikou převádění vět přirozeného jazyka do jazyka logiky alespoň částečně seznámení. Tedy alespoň do takové úrovně složitosti, do níž budou spadat všechny ty příklady, od kterých se bude očekávat řešení. Jistě takto můžeme na základě cílů výuky určit, jaký druh chyb se nesmí v řešení objevit (tedy taková formule, která je velmi špatně utvořená) a hájit se samozřejmě tím, že tyto chyby jsou závažné a nepřipustitelné. Jak ale určit, který druh chyb je vážený více a který méně? Odpovědí by mohlo být asi více, přesto je zřejmé, že chyby by se měly analyzovat (tedy určovat jejich závažnost) na základě toho, kde vznikají nejčastěji. Kterých se nejvíce dopouštějí lidé obeznámeni se základní problematikou. Na následujícím obrázku ještě ujasníme, v jakých případech a co má smysl analyzovat.



Obrázek 16: Vymezení podmnožiny komunity obeznámených se základními principy logiky

Dejme tomu, že A je množina všech lidí na světě, její podmnožina A2 je pak taková množina, která obsahuje všechny ty lidi, kteří jsou seznámeni se základními empirickými poznatky týkající se formalizací vět přirozeného jazyka do jazyka matematické logiky. Dále předpokládejme, že množina B je množinou úplně všech možných zápisů (nebo spíše pokusů) řešení od všech lidí na světě. Její podmnožina B2 je pak množinou všech takových řešení, které dodržují alespoň základní principy sestavování vět prostřednictvím predikátových symbolů, funkčních symbolů, logických spojek, kvantifikátorů, proměnných, konstant a závorek. Jestliže se budeme pohybovat pouze v rámci podmnožin A2 a B2, je teoreticky možné sestrojit solidní mechanismus, který dokáže vyhodnotit studentovo řešení ne pouze jednoznačně shodné s předlohou (jednoznačně správným řešením), ale dokáže také vyhodnotit, na kolik je jeho řešení správné.

Bylo už zmíněno, že zřejmě nejlepším přístupem k tomu, abychom mohli vytvořit mechanismus schopný analyzovat dobře (a částečně „méně“ dobře) utvořené formule je zaměřit se na ta místa, která dělají studentům logiky největší problémy. Ta místa, kde se nejvíce pletou, kde dochází k největšímu počtu chyb. Touto otázkou se už však jednou kdosi zabýval.

3.2. Empirická studie chyb vzniklých při překladu vět z přirozeného jazyka do jazyka logiky

V následující části textu až do konce kapitoly vycházím z empirické studie podle [7].

Jistá skupinka univerzitních vyučujících, konkrétně Dave Barker-Plummer, Robert Dale, Richard Cox, John Etchemendy se rozhodli, že se pokusí zdokonalit výuku logiky za pomocí empirické studie o chybování studentů matematické logiky.

Na základě jejich dlouholetých zkušeností se shodli v tom, že to, na kolik bude studentovo řešení správné, závisí (mimo jiné) především na tom, jak je věta napsána (složena) v přirozeném jazyce. Tedy nakolik bude jednoznačná. Tento fakt je klíčový. Jakmile totiž jednoznačně dokážeme prezentovat nějaké tvrzení v přirozeném jazyce, eliminujeme tak nejednoznačnost při formalizaci tohoto tvrzení do jazyka PL1.

Studie pojednává o vybudování taxonomie typů chyb, které vznikaly u studentů nejčastěji. Základem této studie byl bohatý datový soubor skládající se z různých řešení studentů po celém světě. Na základě těchto výsledků rozdělili chybování do 3 hlavních kategorií. V první kategorii to byly převážně chyby, které se týkaly samotné konstrukce (zaměňování nutných a postačujících podmínek, nerozlišení ekvivalence od implikace, zbytečná složitost apod.). V druhé kategorii jsou chyby, které byly spojeny s pořadím a spojováním slov při překladu. Třetí kategorii pak tvořily chyby pojmenování a zaměňování konstant. Ve všech kategoriích hrálo velkou roli právě jednoznačnost zadání. Tedy čím méně konkrétní, tím více chyb vznikalo. Tvůrci záměrně dávali nelineárně jednoznačné zadání tak, aby se bralo v potaz co nejvíce faktorů. Na druhou stranu, v potaz byly brány některé chyby, které neměly ani tolik co do činění s citem pro logiku, jako spíš nepozornost. Když byly v nějakém příkladu dány konstanty *a*, *b*, *c*, *e*; studenti velmi často chybovali v jejich řešení tím, že zaměňovali ‚*e*‘ za ‚*d*‘, což se nedá považovat za logickou chybu. Nicméně tato skutečnost jenom potvrzuje, jak důležité je studenta seznámit se zadáním tak, aby bylo pro něj co nejjasnější a aby se především na povrch dostávaly jeho přednosti při schopnosti formalizovat přirozený jazyk do jazyka PL1 namísto chytáků a nejednoznačnosti v zadání.

Problémy při formalizaci jsou (přinejmenším částečně) dány charakteristikou překládaného jazyka samotného. Mohli bychom například očekávat, že půjde relativně snadno přeložit větu z přirozeného jazyka na formální tvar, kde mapování z přirozeného jazyka do logických formulí je dáno na základě logických spojek, které představují samostatná slova ve větě v přirozeném jazyce,

jako je například ve větě pro spojku ‚a' obecně užívaná logická spojka konjunkce užívaná v logice. V horším případě však na první pohled dané tvrzení žádné logické spojky neobsahuje.

Ať už je realita jakákoliv, studie pojednávala na základě empirického zjišťování chyb studentů, i to bylo však omezeno počtem zadání. Není tedy možné jednoznačně určit, které lingvistické úkazy jsou pro člověka více problematické a které méně. Studie představuje pouze výsledky z velkého datového souboru subjektů v oblasti logiky 1. řádu.

Výsledky byly získávány na základě balíčku výukového softwaru LPL[3], který se skládal z aplikací[4] (které studenti využívali k vytvoření svého řešení) a uživatelské příručky. Studenti přikládali odpovědi k 748 možných typů cvičení z LPL, které byly následně vyhodnocovány robustním automatizovaným systémem. Výsledky byly pořízeny na bezmála 1.8 milionů poslaných prací od více než 38000 studentů za posledních 8 let. Data byla koncipována z přibližně jednoho sta různých institucí ve více než dvanácti zemích světa. Právě množství těchto prací se dá považovat za relativně odpovídající uvažování lidí při formalizaci a poslouží k lepšímu (přibližnému) pochopení procesů během formálního usuzování.

Porozumění povahy studentských chyb je centrálním pilířem této práce a jinak než postupným zkoumáním velkého množství dat a jeho vyhodnocováním to přesně určit nejde. Vysokofrekvenčními chybami byly především záměny nutné a postačující podmínky, nahrazování logických spojek za jiné a nahrazování konstant za jiné. Tyto poznatky samy o sobě můžou pomoci k obecnějšímu přístupu vyučování logiky.

Celý systém byl uzpůsoben tak, že byla předkládána studentům zadání, která byla sice co nejjednodušší, ale na druhou stranu bylo těchto zadání velké množství s tím, že každé zadání bylo přednostně určeno na to, aby se zaměřilo na jednu konkrétní problematiku (např. zaměňování nutné a postačující podmínky, dosazení konstant apod.). Systém pak na jednoduchých příkladech vyhodnocoval pouze ekvivalenci studentova řešení s některým z množiny vzorových řešení. Při vyhodnocování však student podrobněji nezjistil, v čem chyboval, akorát mu bylo oznámeno sdělení, že „jeho řešení nebylo ekvivalentní žádnému z očekávatelných překladů". Informace týkající se vzorových řešení lze nalézt také na webových stránkách[5].

V celé studii bylo použito mnoho příkladů a každý z nich byl zaměřen na konkrétní problém. Jak bylo řečeno dříve, příklady byly většinou jednoduché a zaměřovaly se na konkrétní problematiky. U nejednoznačných zadání byla vytvořena množina vzorových řešení, které braly v potaz i špatné formalizace. Tyto poté sloužili k snadnější specifikaci a zařazení konkrétní chyby do dané skupiny. Ze všech příkladů a zaslaných studentských řešení bylo identifikováno celkem 45 druhů chyb. Ty se poté roztřídily do tří kategorií.

1) Konstrukční chyby: chyby sestávající ze špatné konstrukce formule. Tyto chyby např. zahrnovaly záměnu postačující a nutné podmínky u implikace, chybějící závorky nebo také absence minimalizace zápisu formule $((A \vee B) \wedge \neg(A \wedge B)$ na místo $(A \vee B))$.
2) Chyby spojek: zaměňování logických spojek za jiné.
3) Atomické chyby: chyby zahrnující strukturu atomických podformulí dělící se na dva typy:
   a. Predikátové chyby, kdy jeden predikátový symbol byl užit na místo jiného.
   b. Argumentové chyby, kde jeden argument byl užit na místě jiného nebo bylo přítomno špatné množství argumentů.

[3] LPL – Language, Proof and Logic
[4] Více informací na http://lpl.stanford.edu
[5] http://ggww2.stanford.edu/GUS/lpl

Každá z kategorií samozřejmě pokrývá sama o sobě kolekci různých druhů chyb. Nepokrývají však určitě všechny možnosti uvažování studenta, jelikož některé druhy chyb byly zcela ojedinělé a nespecifikovatelné, tudíž se ani do studie nezahrnovaly. Následující tabulka představuje (pouze část) určování chyb při formalizaci jednoho ze zadání.

Zadání: Jestliže je *a* čtyřstěn, potom je před *d*. Predikátový symbol C prezentuje čtyřstěn a P představuje relaci „být před.“[6]

#	Vzorové řešení	Chybné řešení	Typ	Podtyp
1.	$C(a) \supset P(a, d)$	$P(a, d) \supset C(a)$	1	AC
2.	$C(a) \supset P(a, d)$	$P(a, b) \supset C(a)$	1, 3b	AC, nesprávná konstanta
3.	$C(a) \supset P(a, d)$	$C(a) \vee P(a, d)$	2	Disjunkce na místo implikace
4.	$C(a) \supset P(a, d)$	$C(a) \supset Q(a, d)$	3a	Nesprávný predikátový symbol
5.	$C(a) \supset P(a, d)$	$C(a) \supset P(a, b)$	3b	Nesprávná konstanta
6.	$C(a) \supset P(a, d)$	$C(a) \supset P(d)$	3b	Chyba arity

Tabulka 8: Podmnožina definovaných modelů a jejich zařezení do kategorií ke vzorovému příkladu

AC – záměna antecedentu za konsekvent (převrácení postačující a nutné podmínky).

Pozn.: U 4. chybového řešení je záměna predikátového symbolu opravdu jen názorná, jelikož ve skutečnosti se jednalo o příklad zadaný v anglickém jazyce a predikátové symboly nebyly jednoznačně dány tak, jak jsem stanovil dříve (C – čtyřstěn; P – být před).

Ze stanovených tří kategorií bylo vybráno dále 8 konkrétních chyb, které byly blíže analyzovány. Ukázalo se, že u těchto konkrétních druhů byly chyby shodné nejčastěji. Podle vzorových zadání se utvořila tabulka osmi nejfrekventovanějších chyb, které byly pro studenty, kteří udělali chybu shodné (chybné řešení se shodovaly) a k nim procentuální vyjádření kolik z chybných řešení bylo shodných (tedy kolik procent studentů chybovalo stejně).

Chyba	Shodnost chybných řešení
Přehození antecedentu za konsekvent	97%
Špatná konstanta	98%
Nesprávný predikátový symbol	74%
Nesprávná logická spojka	94%
Chyba závorek	76%
Záměna argumentů	66%
Chyba arity	100%
Ostatní chyby	52%

Tabulka 9: Podobnost typů chyb z množiny chybných řešení

Pozn.: Samozřejmě je důležité pořád brát zřetel na to, že zadání byla uzpůsobena tak, aby se zaměřovala na zjišťování konkrétního typu chyby, tedy nemohla nastat žádná kombinovaná chyba na takové úrovni, že by ji nebylo možné automatizovaně vyhodnotit. Z toho také vyplývá, že shodnost chybových řešení dosáhlo tak vysokých hodnot.

Automatizovaný mechanismus na zjišťování chyb a jejich zařazování do kategorií nebyl dokonalý, avšak byl schopen klasifikovat celých 85% řešení úspěšně. Dalších 15% tvořily řešení, které nemohly být hlouběji analyzovány, protože obsahovali podstatně více druhů chyb, než mohla zahrnovat množina vzorových řešení (včetně těch nesprávných). Rozdělením by se možná samotné

[6] Původní zadání bylo: Čtyřstěn = tetrahedron (Tet()); je před = in front of (FrontOf())

zařadily do jednotlivých kategorií, avšak z důvodu velkého množství výskytu různorodých chyb to nebylo možné rozumně analyzovat.

Z toho, co vyšlo, však utvořili poměrně přesné statistiky pro nejčastější chybování studentů, které je vzhledem k tak početnému datovému souboru empiricky velmi přesné a je vhodné proto z těchto statistik vycházet. Následující tabulka ukazuje souhrn vůbec nejčastějšího výskytu druhů chyb nad celým datovým souborem. Tabulka se skládá z názvu chyby, počtu chybných řešení a procentuálního vyjádření chybovosti řešení ke všem nalezeným chybám v řešení v datovém souboru.

#	Typ chyby	Počet chyb	Počet chyb v %
1.	Přehození antecedentu za konsekvent	25084	25,86 %
2.	Záměna ekvivalence za implikaci	17518	18,06 %
3.	Záměna implikace za ekvivalenci	11362	11,71 %
4.	Chybná negace	8954	9,23 %
5.	Nesprávná minimalizace (přílišná složitost)	5422	5,59 %
6.	Špatné ekvivalentní úpravy	4701	4,85 %
7.	Chybný argument	4474	4,61 %
8.	Konjunkce místo implikace	3187	3,29 %
9.	Implikace místo konjunkce	2091	2,16 %
10.	Ekvivalence místo konjunkce	1514	1,56 %

Tabulka 10: Vyhodnocení deseti nejfrekventovanějších typů chyb

Pozn.: Typ chyby 5 - (např. $(A \vee B) \wedge \neg(A \wedge B)$ na místo $(A \vee B)$).

Nejčastější chybou je podle všeho zaměňování postačující a nutné podmínky u formule. Podle studie je tento problém ovlivněn do značné míry tím, jak je původní věta sestavena (tím myšleno, že studenti zachovávají slovosled, aniž by si uvědomili, co je ve větě vlastně postačující a nutnou podmínkou u sestavování formule). Jelikož implikace není spojka komutativní, vzniká u ní mnohem více chyb než například u spojování predikátových symbolů pomocí konjunkce, disjunkce apod.

Náplní této práce není dopodrobna zkoumat jednotlivá řešení studentů z datového souboru, ale vhodným způsobem využít data na základě této studie získaná, která poslouží k dalšímu postupu ve vývoji této práce a to specifikace metriky pro podobnost řešení a určování váhy vzniklých chyb u studentova řešení.

4. Metrika podobnosti

V předchozí kapitole jsme došli ke konkrétním závěrům týkajících se nejčastějších druhů výskytu chyb při formalizaci vět z přirozeného jazyka do jazyka logiky na základě empirické studie o vypracování mnoha řešení studentů logiky z různých částí země.

Studie má pro tuto práci sice nespornou výhodu, ale také nějaké nevýhody. Ve studii nebylo bráno v potaz jiné množství individuí než ta konkrétní zmiňovaná – tedy odpadla veškerá problematika s kvantifikátory, přičemž by se dalo říct, že problémy se správným určením konkrétního kvantifikátoru a přiřazením proměnné k němu určené jsou také důležitou částí při vytváření metriky pro podobnost vzorového řešení se studentovým.

Stanovme si (defaultní) váhu jednotlivých chyb (založené na studii z předchozí kapitoly, avšak rozšířené o několik dalších elementů), které budou brány v potaz při analýze studentova řešení. Vyjádříme si je v následující tabulce (Váha chyby se pohybuje na intervalu <0,1>, kde 0 = žádná chyba; 1 = nepřipustitelná chyba) :

#	Typ chyby	Váha chyby
1.	Přehození antecedentu za konsekvent	0.4
2.	Záměna implikace za ekvivalenci	0.4
3.	Záměna implikace za konjunkci	0.4
4.	Záměna implikace za disjunkci	0.5
5.	Záměna ekvivalence za konjunkci	0.5
6.	Záměna ekvivalence za disjunkci	0.5
7.	Záměna disjunkce za konjunkci	0.5
8.	Chybná negace	0.2
9.	Chybný kvantifikátor	0.4
10.	Chybný počet parametrů u predikátu	0.4
11.	Chybný počet parametrů u funkce	0.3
12.	Chybná funkce	0.4
13.	Chybný predikát	0.5
14.	Záměna funkce za proměnnou (konstantu)	0.3
15.	Chybný parametr (proměnná, konstanta) u predikátu	0.2
16.	Přehozené parametry funkce	0.1
17.	Přehozené parametry predikátu	0.1
18.	Chybný parametr ve funkci	0.1
19.	Kritická chyba	1.0

Tabulka 11: Návrh metriky všech možných chyb při porovnávání dvou formulí

Pouze zkráceně uvedu vysvětlivky a příklady k jednotlivým bodům:

1. Jedná se o přehození nutné a postačující podmínky u implikace.

   $(P(x) \supset Q(y))$ namísto $(Q(y) \supset P(x))$

2. Záměna implikace za ekvivalenci nebo ekvivalence za implikaci.

   $(P(x) \equiv Q(y))$ namísto $(P(x) \supset Q(y))$ nebo $(P(x) \supset Q(y))$ namísto $(P(x) \equiv Q(y))$

3. Záměna implikace za konjunkci nebo konjunkce za implikaci.

$(P(x) \wedge Q(y))$ namísto $(P(x) \supset Q(y))$ nebo $(P(x) \supset Q(y))$ namísto $(P(x) \wedge Q(y))$

4. Záměna implikace za disjunkci nebo disjunkce za implikaci.

   $(P(x) \vee Q(y))$ namísto $(P(x) \supset Q(y))$ nebo $(P(x) \supset Q(y))$ namísto $(P(x) \vee Q(y))$

5. Záměna ekvivalence za konjunkci nebo konjunkce za ekvivalenci.

   $(P(x) \wedge Q(y))$ namísto $(P(x) \equiv Q(y))$ nebo $(P(x) \equiv Q(y))$ namísto $(P(x) \wedge Q(y))$

6. Záměna ekvivalence za disjunkci nebo disjunkce za ekvivalenci.

   $(P(x) \vee Q(y))$ namísto $(P(x) \equiv Q(y))$ nebo $(P(x) \equiv Q(y))$ namísto $(P(x) \vee Q(y))$

7. Záměna disjunkce za konjunkci nebo konjunkce za disjunkci.

   $(P(x) \wedge Q(y))$ namísto $(P(x) \vee Q(y))$ nebo $(P(x) \vee Q(y))$ namísto $(P(x) \wedge Q(y))$

8. Chybná negace se vztahuje jak k atomickým formulím tak ke složeným.

   $P(x)$ namísto $\neg P(x)$ nebo $\neg P(x)$ namísto $P(x)$
   $(P(x) \wedge Q(y))$ namísto $\neg(P(x) \wedge Q(y))$ nebo $\neg(P(x) \wedge Q(y))$ namísto $(P(x) \wedge Q(y))$

9. Chybný kvantifikátor se nevztahuje na proměnnou k němu vázanou, pouze na samotný symbol pro kvantifikátor.

   $\exists x$ namísto $\forall x$ nebo $\forall x$ namísto $\exists x$

10. Chybný počet parametrů u predikátového symbolu.

    $P(x, y)$ namísto $P(x)$ nebo $P(x)$ namísto $P(x, y)$

11. Chybný počet parametrů u funkčního symbolu

    $f(x, y)$ namísto $f(x)$ nebo $f(x)$ namísto $f(x, y)$

12. Chybná funkce. Chyba nastane v případě, že jsou všechny parametry funkce špatně zvolené.

    $f(x, y)$ namísto $f(a, z)$ nebo $f(a, z)$ namísto $f(x, y)$

13. Chybný predikát. Tato chyba nastane v případě, že jsou všechny parametry predikátového symbolu špatně zvolené.

    $P(x, f(y))$ namísto $P(a, z)$ nebo $P(a, z)$ namísto $P(x, f(y))$

14. Záměna funkce za proměnnou nebo konstantu. Tato chyba může nastat pouze u ověřování parametrů predikátového symbolu. Je více vážená, než kdyby nastala záměna proměnné za jinou proměnnou (konstantu) nebo naopak.

    $P(f(y))$ namísto $P(y)$ nebo $P(y)$ namísto $P(f(y))$

15. Chybná proměnná (resp. konstanta) u predikátového symbolu - méně vážená než předchozí.

P(x) namísto P(y) nebo P(y) namísto P(x)

16. Přehozené parametry u funkce.

    f(x, y) namísto f(y, x) nebo f(y, x) namísto f(x, y)

17. Přehozené parametry u predikátového symbolu.

    P(x, f(y)) namísto P(f(y), x) nebo P(f(y), x) namísto P(x, f(y))

18. Chybný parametr ve funkci je vlastně špatně zvolená proměnná (resp. konstanta) u funkčního symbolu.

    f(x) namísto f(a) nebo f(a) namísto f(x)

19. Kritická chyba může nastat v případě špatné konstrukce formule, v nedodržování povolených symbolů apod.

Schopnost zachytit, vyhodnotit a správně určit na kolik je formule správně zapsaná, jestliže obsahuje více než jednu chybu je značně složité. Jediným aplikovatelným východiskem je trvat na tom, aby věty v zadání byly sepsány tak, aby dávaly co nejjednoznačnější smysl a tedy tím zajistit, aby se nemuselo brát v potaz tolik možností správného zápisu studentova řešení.

Náplní této práce je vytvořit takový algoritmus, který na základě jednoho správného vzoru bude porovnávat řešení studenta do jaké míry je jeho řešení správné. V prvním kroku se ověří přímo, jestli je studentovo řešení ekvivalentní tomu vzorovému. Jestliže tomu tak nebude, podstoupí toto řešení analýze, která určí, jaký druh chyby se v řešení vyskytoval a na základě statistik nejfrekventovanějších chyb dále určí, na kolik je toto řešení správné (za jednotlivé chyby se bude strhávat část bodů (procentuelně) a ve výsledku se vrátí adekvátní bodové ohodnocení vypracovaného řešení). Krom toho se bude zaznamenávat počet zachycených chyb.

Jednoznačnost zadání je nutnou podmínkou k tomu, aby mohla analýza probíhat bez neočekávatelných událostí. Jedním z faktorů pro upřesnění zadání bude fakt, že studentovi bude k zadání připojena tabulka, která bude obsahovat pojmenování všech elementů vystupujících ve formuli. Tímto odpadne problém samovolného pojmenovávání predikátových a funkčních symbolů, stejně jako pojmenovávání proměnných a konstant. Algoritmus se pak bude odkazovat na konečnou množinu symbolů, které mohou být použity v řešení. V následující tabulce je výčet všech možných symbolů, které budou akceptovány při zpracování vzorového a studentova řešení.

Skupina	Prvky	Vysvětlivky
Predikátové symboly	P; Q; R	Představují jednotlivé predikátové symboly
Funkční symboly	f; g; h	Představují jednotlivé funkční symboly
Proměnné	x, y, z	Názvy proměnných
Konstanty	a, b, c	Názvy konstant
Logické spojky	¬, *; +; >; =	> - implikace; + - disjunkce; * - konjunkce, = - ekvivalence; ! - negace
Kvantifikátory	1; !	1 – existenční; V – všeobecný
Závorky	(;)	Závorky

Tabulka 12: Výčet všech povolených symbolů použitých při stavbě logických formulí

Mimo jiné je potřeba počítat i s bílými znaky a také symboly pro oddělení parametrů u predikátových (resp. funkčních) symbolů. Pokud vymezíme tuto podmnožinu pro všechna možná přípustná řešení, značně se nám usnadní práce. Mimo jiné můžeme spoléhat na to, že se všechna řešení budou držet standardních principů sestavování formulí PL1 a jelikož je gramatika pro sestavování formulí z přirozeného jazyka do jazyka PL1 konečná a v podstatě pevně daná, rovněž se lze těmito pravidly řídit při ošetřování samotné formule. Gramatika pro formalizaci vět z přirozeného jazyka do jazyka PL1 by mohla vypadat například takto:

Gramatika	Vysvětlivky
S => KF	*S – startovní neterminál*
K => C | CC | CCC	*K – kvantifikátory*
C => !Prom | 1Prom	*C –vázání kvant. k proměnné*
F => (ABA) | A	*F – formule*
A => (A2BA2)	*A – podformule*
A2 => Pred(P0) | Pred(P0,P0) | ¬Pred(P0) | ¬Pred(P0,P0)	*A2 – predikátové symboly*
B => > | + | * | =	*B – logická spojka*
Pred => P | Q | R	*Pred – predikátový symbol*
Prom => x | y | z	*Prom – proměnná*
Kon => a | b | c	*Kon – konstanta*
P0 => P1 | f(P1) | f(P1,P1) | g(P1) | g(P1,P1) | h(P1) | h(P1,P1)	*P0– parametr pred. symbolu*
P1 => Prom | Kon	*P1 – proměnná či konstanta*

Tabulka 13: Gramatika sestavující logické formule

V případě návrhu gramatiky jsem vycházel z [8].

4.1. Základní předpoklady

Základními předpoklady pro bezproblémový průběh analýzy je dodržovat pokyny u slovního zadání. Metrika je vymezená (testově) pouze na symboly, které jsou uvedeny v tabulce povolených symbolů. Jestliže tyto nebudou dodržovány a řešitel si bude volit své stavební prvky sám, algoritmus vyhodnotí jeho řešení jako konstrukčně chybné. Dalším předpokladem je, že samotné vzorové zadání může nabývat takové formy, jakou udává uvedená gramatika. Respektive každá formule bude rozdělena na 2 části tak, že vlevo budou umístěny kvantifikátory, vpravo samotná formule (jak udává gramatika).

Algoritmus dále přísně zohledňuje to, jestli se počet levých závorek rovná počtu pravých závorek. Ošetření a automatizované doplnění závorek algoritmem na konkrétní místa tam, kde by je teoreticky student mohl položit, není příliš rozumné. V algoritmu musí počet pravých i levých závorek odpovídat, jelikož se nad nimi provádějí v průběhu výpočty a rozlišuje se, jestli jsou závorky vázané k predikátovým (resp. funkčním) symbolům anebo oddělují samotnou logiku sestavování vět. Jedním z možných řešení je studenta při psaní svého řešení a následného pokusu o odeslání upozornit, že počet jeho levých závorek neodpovídá počtu pravých (jedná se opravdu o chybu z nepozornosti, jelikož každý ví, že závorky vymezují určitý obsah a tedy se tento musí nacházet právě mezi levou a pravou závorkou. Ani při ručním vyhodnocování by takovou chybu nebral vyučující jako opravdovou chybu). Následující část kódu ukazuje, jak by mohl být tento problém ošetřen:

```
for (int i = 0; i < reseniStudentovo.Length; i++)
            {
                if (reseni[i] == '(') levaZavorka++;
                if (reseni[i] == ')') pravaZavorka++;
            }
```

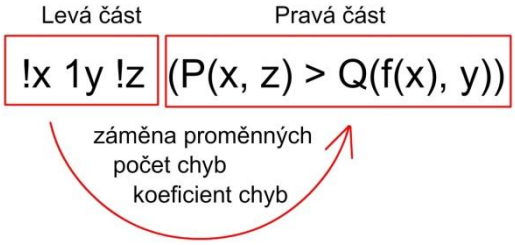
```
if (levaZavorka != pravaZavorka)
{
Console.WriteLine("levých závorek je v řešení: " + levaZavorka
+ " a pravých: " + pravaZavorka);
} else  instance.odesliReseni(reseniStudentovo);
```

Algoritmus dále nebere v potaz zákony pro převody logických spojek, De-morganovy zákony, zákony pro implikaci. Možným východiskem je stanovit více vzorových vypracování a jednotlivé vzory pak pomocí algoritmu porovnávat se studentovým řešením, případně postupem času vytvořit modul, který bude na základě rozparsování generovat všechny možné ekvivalentní zápisy dané formule.

4.2. Ošetření kvantifikátorů

Před samotným ošetřením formule je dobré nejdříve zhodnotit kvantifikátory. V podstatě při každém přepisu z přirozeného jazyka do PL1 začínáme od leva (od kvantifikátorů). Kvantifikátory nám slouží k vyjádření míry přítomnosti predikátových symbolů v jisté třídě objektů. To je jejich význam, a proto je největší důraz brán na to, jaký kvantifikátor byl zvolen. Tedy zásadní chybou při volbě kvantifikátoru je jeho špatná volba. V takovém případě je to bráno za chybu a postupuje se bez další inkrementace koeficientu pro sčítání váhy chyby do druhé části (ověřování samostatné formule). Jestliže jsou však kvantifikátory zachovány ve svém pořadí tak, jak mají být, pořád existuje spousta možností, jak kvantifikátory ekvivalentně zapsat. Ke každému kvantifikátoru se váže konkrétní proměnná. Ta musí být samozřejmě pro každý kvantifikátor odlišná. Jestliže se nachází ve formuli 2 (resp.3) shodné kvantifikátory, je pak možné libovolně prohazovat mezi sebou proměnné k nim vázané, aniž by to bylo považováno za chybu.

Dále pak je v algoritmu ošetřována situace, kdy kvantifikátory jsou zvoleny správně, avšak proměnné neodpovídají (tedy že jsou proměnné u nich prohozeny tak, jak to povoleno není. Např. záměna proměnné u všeobecného kvantifikátoru za proměnnou u existenčního). Tato situace není brána za chybu, avšak algoritmus si zapamatuje, o jaké záměny se jednalo a na jejich základě pak následně při vyhodnocování formule bude brán zřetel – zadání se uzpůsobí těmto záměnám. Jako menší chyby pak jsou brány zapomenutí negace či její nadbytečnost u daného kvantifikátoru. Celá část pro kvantifikátory ověřuje velké množství možností, které mohou nastat, avšak svým návrhem pevně stanovuje hranice, kam je povoleno vstoupit tak, aby se nejednalo o chybu.



Obrázek 17: Předávané parametry po vyhodnocení levé části formule s kvantifikátory

V prvním kroku se nejdříve ověří celá formule, zdali neobsahuje znaky, které nemají ve formuli co dělat (bílé znaky se samozřejmě ignorují). Jestliže je formule složena pouze ze stavebních kamenů, které jsou povoleny, může začít samotný proces ověřování kvantifikátorů.

1. Vytvoří se tabulka reprezentující všechny možnosti toho, co se může objevit v levé části formule (tam, kde je vymezený prostor pro kvantifikátory). Pro jednu proměnnou to můžou být symboly negace, 2 druhy kvantifikátorů (všeobecný anebo existenční) a 3 typy proměnných (x anebo y anebo z). Jelikož se celá gramatika vztahujeme k nejvýše třem

proměnným, tabulka může obsahovat maximálně 18 prvků. Následující obrázek snad lépe ujasní můj výklad.

Pozice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	¬	∀	∃	*x*	*y*	*z*	¬	∀	∃	*x*	*y*	*z*	¬	∀	∃	*x*	*y*	*z*
	0	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0

Obrázek 18: Tabulka zachycující přítomnost povolených prvků v levé části formule

Příznak 1 u daného symbolu znamená, že se ve formuli tento znak nachází. Podle obrázku bude levá strana formule vypadat takto: ∀*x* ∃y.

Tabulky budou ve skutečnosti 2. Jedna pro vzor, druhá pro řešení.

2. Po naplnění tabulek se obě porovnávají a vyhodnocují se další omezení. Kritickou chybou je rozdílný počet použitých proměnných a nebo také vícenásobné použití stejné proměnné. Na základě pamatování počtu použitých proměnných mohou nastat 3 případy.

   a) Formule obsahuje jediný kvantifikátor s jedinou proměnnou. Vzor i řešení se vztahují pouze ke všem možnostem kombinací v tabulce od 0 do 5. Ověří se negace, kvantifikátory a nakonec proměnné. Jestliže dojde k záměně proměnných (avšak kvantifikátory jsou určeny dobře), zaznamená se tato změna do další tabulky. Ta bude vyhodnocována až při vyhodnocování samotného těla (pravé části) formule už bez kvantifikátorů.

   b) Formule obsahuje 2 kvantifikátory se dvěmi proměnnými. Vzor i řešení se vztahují pouze ke všem možnostem kombinací v tabulce od 0 do 11. Zde počet možností roste. Proto se zde ověřuje i to, jestli jsou oba kvantifikátory shodné či nikoliv. V prvním případě nezáleží na tom jak jsou proměnné uspořádané, avšak jestliže je ve vzoru u všeobecných (resp. existenčních) kvantifikátorů použito proměnných např. *x* a *y* a v řešení je např. *y* a *z*, pak je i toto nutno zaznamenat do další tabulky pro záměnu proměnných.

   c) Formule obsahuje 3 kvantifikátory se třemi proměnnými. Vzor i řešení se vztahují k celé tabulce (0 – 17). Počet možností je na první pohled ještě vyšší, naštěstí v tomto případě můžeme s jistotou tvrdit, že zde budou minimálně 2 kvantifikátory, které se opakují, tedy záměna proměnných k nim vázaných není chyba, avšak je důležité pořád dodržovat správné proměnné. Jestliže toto není dáno, pak se opět zapisují záměny do tabulky pro záměnu proměnných. V lepším případě budou všechny 3 kvantifikátory shodné – všechny tři všeobecné(resp. existenční). V takové případě se mohou proměnné libovolně zaměňovat a jelikož máme z předchozích předpokladů zaručeno, že se v řešení nebudou vyskytovat jiné znaky než ty, které jsou dány gramatikou, nemůže nastat žádná nepředpokládaná záměna proměnných. Jestliže nebudou všechny symboly pro kvantifikátory shodné, znamená to, že pouze jeden z nich je jiný a jelikož musí být ke třem symbolům pro kvantifikátory využity všechny tři proměnné, ověřuje se pouze ta proměnná, která já vázána k tomu kvantifikátoru, který je z přítomných tří unikátní.

Tabulka záměny proměnných je pevně daná tabulka o 6ti prvcích , kde každý prvek představuje konkrétní záměnu. Tato záměna je následně identifikovaná v této tabulce přidáním příznaku – jedničky – na konkrétní pozici. Pozice [0] prezentuje záměnu *x* na *y* (jestliže je ve vzoru použito *x* a student použil *y*). Pozice [1] prezentuje záměnu *x* na *z*. Pozice [2] prezentuje záměnu *y* na

x. Pozice [3] prezentuje záměnu *y* na *z*. Pozice [4] prezentuje záměnu *z* na *x*. Pozice [5] prezentuje záměnu *z* na *y*. Tato tabulka bude ještě důkladněji popsaná (a zobrazena) v ošetřování pravé části (formule bez kvantifikátorů).

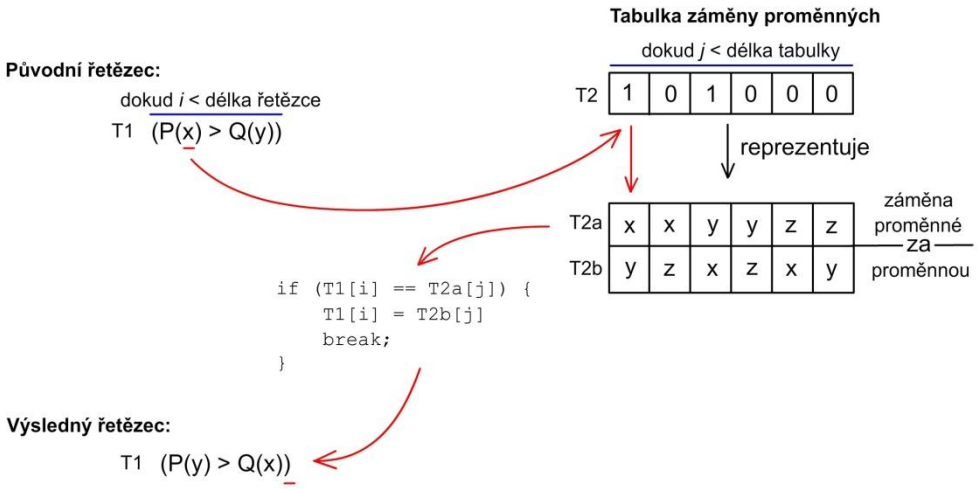
Před samotným vstupem do formule se ještě ověří, jestli není před formulí (ne před kvantifikátorem) negace.

4.3. Ošetření formule

Po vyhodnocení první části s kvantifikátory se vstupuje do druhé (pravé) části, kde se ošetřuje samotná logická formule, přičemž se z předchozí části předávají parametry určující, kolik chyb bylo v části s kvantifkátory provedeno, jakou váhu dohromady čítají a taky to, jestli došlo k záměně proměnných a jak tato záměna vypadá. Na začátku jsou kromě těchto parametrů předávány dva řetězce – pro vzor a řešení a k nim pozice začátku samotné formule pro obě varianty (tedy to místo v řetězci, kde končí vše spjaté s kvantifikátory a začíná samotná formule).

V prvním kroku se díky zjištěným pozicím počátku samotné formule upraví řetězce tak, že se z nich odstraní prefix, který obsahuje kvantifikátory a k nim vázané proměnné. Tím se získá čistá formule. Z té se poté odstraní všechny bílé znaky a následně dojde k procesu úprav všech proměnných v samotné formuli podle toho, jestli bylo u předchozího kroku (ošetřování kvantifikátorů) zaznamenáno, že došlo k záměně proměnných. Samotný postup bude vypadat následovně:

Řetězec pro vzorové řešení si rozparsujem na jednotlivé znaky a uložíme je do pole (pojmenujme Pvzor). Toto pole následně v cyklu procházíme a hledáme nějakou z proměnných. Jestliže jsme našli některou z proměnných na pozici *i*, začneme v cyklu procházet druhé pole – tabulku záměny proměnných (zmiňována výše – tabulka o 6ti prvcích reprezentující všechny možné záměny proměnných). Jestliže v tabulce Záměny proměnných narazíme na příznak 1 na pozici *j* (tedy jsme došli ke konkrétní záměně) a zároveň nalezený znak proměnné v poli Pvzor na pozici *i* je roven znaku, který je vázán k tabulce záměn na pozici *j*, pak do pole Pvzor na pozici *i* uloží novou proměnnou, která je vázána na nalezenou shodnou proměnnou na pozici *j*. Poté ihned vyskočí z cyklu, který prohledává tabulku záměny proměnných a prochází dále pole Pvzor. Vyskočit z cyklu ihned po uložení nové proměnné do tabulky Pvzor je nutné, jelikož po přepsání by se procházela tabulka záměn dále a mohlo by dojít k opětovnému přepsání proměnné na původní (jestliže přepíše proměnnou *x* na *y* a nevyskočí hned z cyklu, na další pozici by program identifikoval proměnnou *y* a tu přepsal zpět na proměnnou *x*). Následující obrázek vše lépe vysvětlí.



Obrázek 19: Princip náhrady proměnných v logické formuli

Po dokončení procesu záměny proměnných opět tabulku Pvzor převedeme na řetězec. Ještě doplním, že úprava proměnných v řetězci se nekoná nad studentovým řešením, ale nad vzorovým. To se snaží studentovu řešení pokud možno vždycky přiblížit.

Po této části už jsou oba řetězce zkontrolovány, jestli neobsahují zakázané symboly, jsou zbaveny bílých znaků a také jsou provedeny všechny ekvivalentní úpravy v případě, že u kvantifikátorů došlo k záměně proměnných. V další části se bude konrolovat konstrukce této formule. V této části se kromě počtu chyb a jejich ohodnocení předávají pouze řetězce obsahující upravený vzor a řešení. V prvním kroku se v obou formulích zjistí počet predikátových symbolů a ty se následně porovnávají. V případě neshody počtu predikátových symbolů je toto považováno za konstrukční chybu, avšak v takovém případě probíhá analýza formule vždy až do konce. Stejně jako počet predikátových symbolů se tento počet zjišťuje i pro logické spojky. Rovněž v případě neshody je toto považováno za konstrukční chybu, ale pokračuje se dál (jak už bylo řečeno dříve, tato práce nezohledňuje různé zákony pro ekvivalentní převody formulí, jelikož by se tím celá práce velmi zkomplikovala).

Původně se zjišťovali ve formulích i počty funkčních symbolů, avšak považovat za velkou chybu neshodu v jejich počtu není na místě, protože podle metriky je brán zřetel na špatně zvolený parametr u predikátového symbolu a i když je toto považováno za chybu, není to chyba až tolik vážená, jako například chybný počet predikátových symbolů u vzoru a řešení.

Jestliže je počet predikátových symbolů u vzoru i řešení roven jedné a počet logických spojek tedy logicky roven 0 (počet logických spojek je roven počtu predikátových symbolů mínus jedna), znamená to, že formule je triviální, není třeba jí dále dělit a rovnou se volá funkce pro ověřování atomických predikátových symbolů[7] (která bude popsána později).

Po předchozích ověřeních se volá funkce, která z obou formulí po průchodu vrátí pozici hlavní spojky, od které je poté možné rozdělit každou z formulí na dvě části. Tato funkce právě využívá toho předpokladu, že počet levých závorek ve formuli je roven počtu pravých.
Jak algoritmus pracuje vysvětlí následující popis.

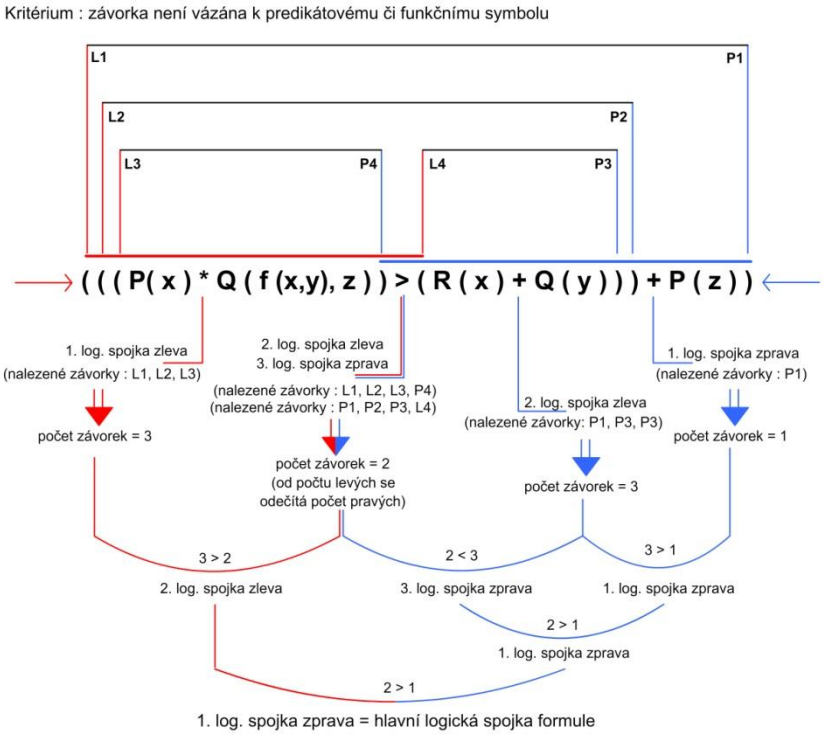
V prvním kroku zjistí počet logických spojek a to tak, že prochází řetězec a jakmile nalezne jednu ze symbolů pro logické spojky, inkrementuje proměnnou zastupující tento počet a do proměnné *pozice* ukládá (přepisuje) číslo pozice, na kterém byla logická spojka nalezena. Jestliže je po průchodu celého řetězce počet logických spojek roven jedné, vrátí funkce jednoduše proměnnou *pozice* (více logických spojek nalezeno nebylo). Jestliže bylo nalezeno spojek více, pak se *pozice* vynuluje a postupuje se podle dalšího scénáře.

V prvním kroku se řetězec prohledává zleva a hledá všechny ty levé závorky, které nejsou spjaty s žádným predikátovým symbolem, tzn. závorky vázané k predikátu, ale i závorky vázané k funkci uvnitř predikátu (tedy jsou to závorky upřesňující, která logická spojka má větší prioritu). Jakmile narazí na poslední takovou (levou) závorku, která splňuje kritéria (že není vázána k predikátovému či funkčnímu symbolu), uloží si tuto pozici. Nyní je jasné že hlavní logická spojka se může nacházet v levé části řetězce někde na pozici 0 až na pozici poslední levé závorky splňující kritéria. Následně se sekvence (řetězec) prohledává opět zleva (od 0 až po pozici poslední levé závorky splňující kritéria) a zaznamenávají se opět všechny levé závorky takové, které nejsou žádným způsobem spjaty s predikátovými symboly. V případě nalezení takové levé závorky se počet těchto závorek inkrementuje. Zároveň se však zazanamenává počet pravých závorek splňující opět stejné

[7] Atomickým predikátovým symbolem nazývám vlastně pouze atomickou formuli ve tvaru Predikát(term1) případně Predikát(term1, term2)

podmínky (tedy to, že nejsou vázány k žádnému predikátovému nebo funkčnímu symbolu) a jestliže se taková závorka najde, pak se počet levých závorek dekrementuje. Zároveň ověřujeme, jestli se někde na této podsekvenci (0 až pozice poslední levé závorky splňující kritéria) nenachází nějaká logická spojka. Jestliže je nalezena, pak je tato spojka kandidátem na hlavní spojku. Uložíme si její *pozici* a také počet doposud nalezených levých závorek (samozřejmě jak bylo psáno výše, toto číslo se mění na základě toho, jestli nalezneme levé či pravé závorky splňující dané kritéria - nesmí se vázat k predikátovému nebo funkčnímu symbolu). Jelikož je to první nalezená logická spojka zleva, dalo by se očekávat, že právě ona je ta hlavní (z levé strany sekvence). To ovšem nemusí být vůbec pravda, proto se počet levých závorek nalezených do doby nalezení logické spojky uchová v paměti. Jestliže následně narazíme na další logickou spojku, pak si hodnotu z paměti opět načteme a porovnáme, jestli je větší nebo rovna aktuálnímu počtu levých závorek. Jestli tomu opravdu takto je, pak je nová potenciální hlavní spojka na světě, v paměti se původní hodnota počtu levých závorek přepíše a zaznamená se *pozice* této spojky.

Analogicky se takto postupuje až do konce levé části řetězce (jestliže je nalezena další logická spojka a hodnota levých závorek je shodná, pak se přepíše jenom *pozice* potenciální hlavní spojky, jelikož při formalizaci se obvykle závorkuje zleva). Podobný princip se poté opakuje s tím rozdílem, že se sekvence začíná prohledávat zprava a počítají se pravé závorky. Změnou v této části je, že jelikož postupujeme zprava, tak je pro nás rozdíl, jestli při nalezení další logické spojky a k ní načtení počtu pravých závorek z paměti (předchozí nalezené logické spojky) je tento počet větší než ten dříve zaznamenaný anebo větší nebo roven. V případě prohledávání zprava musí být tento počet pravých závorek načtených z paměti pouze větší než ten aktuálně počítaný - při nalezení další logické spojky (jak už bylo řečeno, závorkuje se zleva, tudíž narozdíl od levé strany, kdy se snažíme přiblížit té pravé, tady jsme prakticky úplně vpravo už od začátku, tedy všechny další nalezené logické spojky se shodným počtem pravých závorek jsou ignorovány. Jako potenciální hlavní logická spojka je uložena ta na *pozici*, kde počet pravých závorek je pouze menší než počet závorek při nalezení předchozí logické spojky – té s nejnižším číslem u proměnné zachovávající počet pravých závorek).



Obrázek 20: Princip zjišťování centrální logické spojky v logické formuli

Po tomto procesu máme k dispozici 2 potenciální hlavní spojky. Nyní už jen porovnáme počty levých a pravých závorek uchovávané k pozicím spojek. Jestliže je počet levých závorek menší než pravých, pak je hlavní spojkou řetězce spojka nalezená v levé části. Jestliže je tomu naopak, pak je hlavní spojkou ta v pravé části. V případě, že se oba počty sobě rovnají, bere se jako hlavní spojka ta v pravé části. Celá funkce pak vrací *pozici* logické spojky.

Na obrázku výše je ukázka, jak princip funguje. V tomto konkrétním případě je hlavní spojkou první spojka nalezena zprava (značena +). Funkce vrací *pozici* této spojky.

Tato funkce je volána pro obě formule – vzor i řešení, z čehož nám vzejdou 2 proměnné uchovávající v sobě *pozici* centrální logické spojky pro vzor a řešení. Zjišťovat hlavní spojku pro obě formule není zbytečné. Nastat může například situace, kdy řešení bude obsahovat o mnoho závorek více než vzor (případně budou v řešení chyby – nadbytečné nebo chybějící negace, chybný počet argumentů u predikátových či funkčních symbolů a podobně, ale to pořád neznamená, že je formule úplně špatně), přesto však bude ekvivalentní vzoru. Čísla udávající pozici hlavní spojky pak budou rozdílná. Pozice hlavní spojky je důležitá pro následné dělení obou formulí na další podsekvence a taky k zjišťování znaku který se ukrývá pod pozicí hlavní spojky. Zjišťování tvaru hlavní logické spojky je vlastně hned dalším krokem, ten je však triviální, jelikož již známe pozici této spojky, není problém zavolat funkci, která nám na základě správně zvoleného řetězce a k němu adekvátního čísla značící *pozici* hlavní spojky vrátí znak reprezentující hlavní logickou spojku. I tato funkce se volá dvakrát – jednou pro vzor a podruhé pro řešení.

Jakmile známe pozice centrálních logických spojek obou formulí, rozdělíme tyto formule na další části – levou stranu od centrální spojky a pravou stranu od centrální spojky. Utváření stran je řešeno voláním dvou funkcí, kde každá z nich vytváří jednu stranu – levou a pravou a to jak pro vzor, tak pro řešení (celkem budou tedy ve výsledku 4 podsekvence).

Nejdříve se volá funkce pro vytvoření levé strany formule. Té se předávají jako parametry celá formule a pozice hlavní logické spojky. Uvnitř funkce poté procházíme zleva formuli znak po znaku až do pozice centrální spojky a znovu počítáme levé a pravé závorky (nyní pro nás ovšem není důležité vědět, která z nich se váže či neváže na predikátové nebo funkční symboly). V nejlepším případě bude počet závorek roven 0. V takovém případě to znamená, že není potřeba odstraňovat žádné přebytečné závorky. Jestliže je počet závorek větší než 0, pak se toto číslo uloží do paměti (a později také předá funkci, která bude konstruovat pravou stranu formule). Když už známe počet (nadbytečných) závorek v levé části, začneme opět zleva procházet řetězec znak po znaku od 0 až do pozice centrální logické spojky (aktuální pozici v sekvenci pojmenujme např. *i*). Hned v první fázi vstoupí program do dalšího cyklu, který bude probíhat do té doby, dokud nebude počet (nadbytečných) závorek roven 0. V cyklu pak ověřuje:

> Jestli se na pozici *i* v řetězci nachází negace, pak inkrementuje *i* o +1.
> Jestli se na pozici *i* v řetězci nachází levá závorka, pak dekrementuje počet závorek o -1 a inkrementuje *i* o +1.

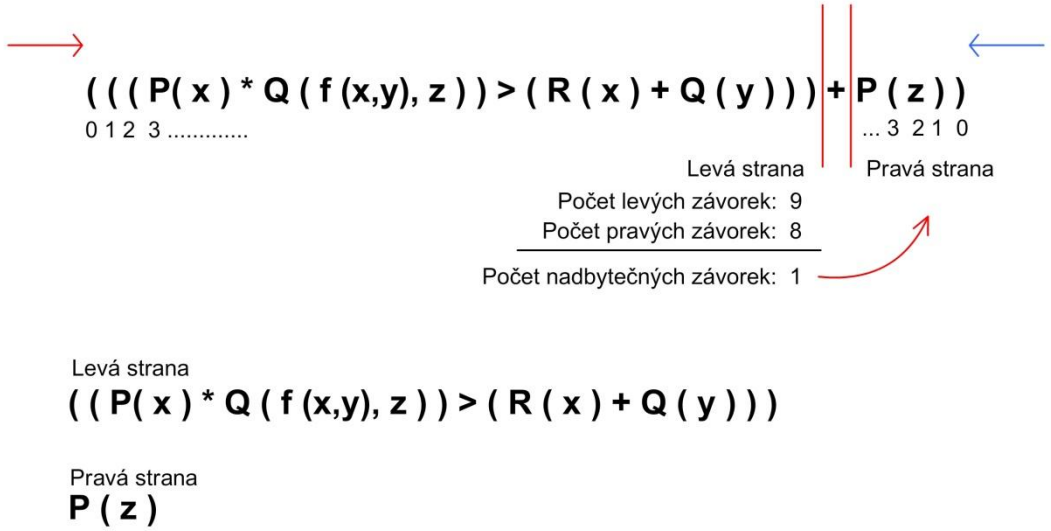
Takto postupuje, dokud není počet závorek roven 0. Poté začne ukládat jednotlivé znaky (od pozice *i* v řetězci) do nového řetězce, představujícího levou stranu od centrální logické spojky, která už neobsahuje nadbytečný počet závorek.

Jakmile vrátí funkce levou stranu řetězce, volá se funkce pro vytvoření pravé strany. Té předává (kromě stejných parametrů, jako při volání funkce na vytvoření levé strany formule) navíc parametr udávající nadbytek závorek. Rozdíl je v tom, že se zde opět prochází formule zprava až do pozice centrální logické spojky. Tedy se prochází znak po znaku od konce řetězce až do pozice hlavní spojky (aktuální pozici v sekvenci pojmenujeme např. *j*). Program opět vstoupí do dalšího cyklu,

který bude probíhat do té doby, dokud nebude počet (nadbytečných) závorek (předaný parametr z předchozího výpočtu u vytváření levé strany formule) roven 0. V cyklu pak ověřuje:

> Jestli se na pozici *j* v řetězci nachází pravá závorka, pak dekrementuje počet závorek o -1 a zároveň dekrementuj *j* o -1.

Takto opět postupuje, dokud není počet nadbytečných závorek roven 0. Teprve pak začne do nově vytvořeného pole postupně ukládat od konce řetězce jednotlivé znaky pro pravou stranu. Nakonec se pole obsahující všechny znaky z pravé strany převede na řetězec prezentující pravou stranu formule od centrální logické spojky, která už neobsahuje nadbytečný počet závorek. Tento řetězec pak funkce vrací.



Obrázek 21: Princip získávání částí při dělení formule na 2 bloky

Po zavolání funkcí pro vytvoření jednotlivých stran formule ještě zavoláme metodu, která nám ze všech čtyř řetězců (levá strana pro vzor, levá strana pro řešení, pravá strana pro vzor a pravá strana pro řešení) spočítá počet predikátových symbolů. Toto číslo využijeme později při zjišťování, zda nejsou jednotlivé strany formule u vzoru a řešení přehozeny.

Jakmile máme pravou i levou stranu pro vzor i řešení, známe pozice hlavních logických spojek i to, jaké logické spojky to jsou a navíc víme, kolik predikátových symbolů se nachází na každé straně, pak nám nic nebrání tomu, abychom volali rekurzivní funkci, která bude dále získané strany dělit na menší podsekvence, odhalovat špatně dosazené negace, rozhodovat o tom, jestli jsou strany přehozené, porovnávat logické spojky a čítat chyby i jejich vážnost.

V první části funkce se budou zjišťovat jednotlivé názvy predikátových symbolů na obou stranách (i obou formulí pro vzor a řešení). Předem získaný počet predikátových symbolů z každé strany nám pomůže vytvořit pole o velikosti rovnající se právě počtu těchto symbolů. Do pole se pak budou ukládat konkrétní znaky reprezentující predikátové symboly.
Jakmile se takové pole vytvoří pro každou stranu, můžou se navzájem tyto pole porovnávat.

1. Jestliže se pole (s predikátovými symboly) pro levou stranu vzoru rovná poli pro levou stranu řešení a zároveň se pravá strana vzoru rovná poli pro pravou stranu řešení, pak jsou uspořádány ve správném pořadí (vždycky se porovnávají 2 pole, které se nejdříve lexikograficky setřídí. To je nutné kvůli tomu, že pole se porovnává po jednotlivých buňkách od začátku až po konec. Jestliže se nacházejí v nějaké podsekvenci formule komutativní

spojky a řešitel tyto strany ekvivalentně zamění, pak to nemůže být bráno za chybu, ale kdyby nedošlo předtím k setřídění pole, pak by program chybu vrátil).

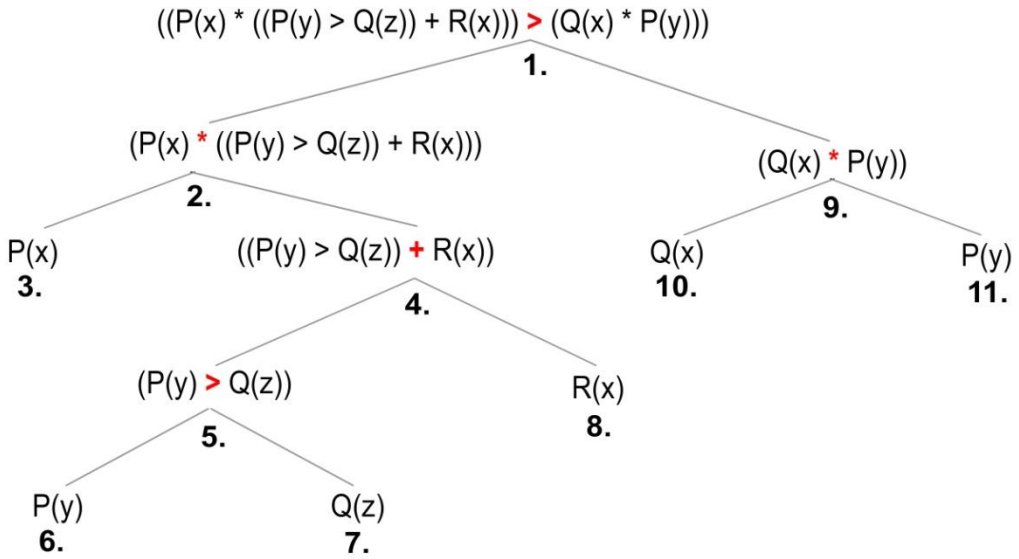
2. Jestliže se pole pro levou stranu vzoru rovná poli pro pravou stranu řešení a pole pro pravou stranu vzoru je rovno poli pro levou stranu řešení, pak jsou tyto strany přehozeny, ale jsou správně (vyhodnocování komutativity spojek přijde až později).
3. Jestliže jsou všechny pole (jejich jejich velikost i všechny prvky) sobě rovné, pak není možné rozlišit, jestli jsou strany přehozeny nebo ne.
4. Jestliže po porovnání neodpovídá výsledek bodům 1 – 3, pak je řešení konstrukčně špatně.

U každé podsekvence se navíc ověřuje správnost negace před konkrétní podformulí (levou a pravou stranou) anebo přímo před atomickým predikátovým symbolem. Toto ověřování je vždycky vázáno na každý z výše zmíněných bodů (jelikož například při přehození stran se musí brát zřetel na to, že negace se bude porovnávat u levé strany vzoru s pravou stranou řešení apod.).

Jestliže jsou strany přehozené, pak se tyto strany přehodí (opět u vzoru, který se snaží přiblížit co nejvíce řešení) tak, aby byly ve správném pořadí.

V dalším kroku se porovná hlavní spojka ve vzoru s hlavní spojkou v řešení. Jestliže se sobě rovnají, pak se ověřuje, co za spojku se to vlastně jedná. Jestliže se jedná o komutativní spojku (konjunkce, disjunkce, ekvivalence), pak se nic nestane (i když byly strany přehozeny). Jestliže se však o komutativní spojku nejedná (v tomto případě jestliže se jedná o implikaci) a strany přehozeny opravdu byly, pak se jedná o chybu a to záměnu antecedentu za konsekvent (přehození nutné a postačující podmínky u implikace).

Jakmile je spojka nalezena (a ošetřena v případě předchozí úvahy o přehození postačující a nutné podmínky u implikace), pak se hlouběji dostáváme do jednotlivých stran obou formulí, kde vždy nejdříve u levé strany obou formulí ověřujeme, jestli obsahuje další spojku. Jestli tomu tak skutečně je, pak se celý proces (nad levými stranami vzoru a řešení) hledání pozice (a reprezentanta) hlavní logické spojky, levých a pravých stran, počtu symbolů vlevo a vpravo, opakuje, načež se rekurzivně opět volá funkce, která získané strany bude dále ověřovat a dělit na další podsekvence. V případě, že už levá strana další logickou spojku neobsahuje, volá se funkce pro ověření atomického predikátového symbolu. Jakmile je celá levá strana ošetřena, pak se stejný proces aplikuje i na pravou stranu, dokud není celá formule ošetřena.



Obrázek 22: Princip sestupu až do nalezení atomického predikátového symbolu

Jednotlivé čísla označují, jak program prochází formulí. Jak je vidět na struktuře stromu, program vždy prochází strom zleva. Když už nemá možnost jít vlevo hlouběji, pak začne ošetřovat atomický predikátový symbol a jakmile s analýzou skončí, tak pokračuje vpravo od dělícího uzlu (opět jestliže už nenalezl v pravé straně další logickou spojku, tak začne ošetřovat predikátový symbol a až skončí, tak pokračuje v analýze pravých stran o úroveň výš).

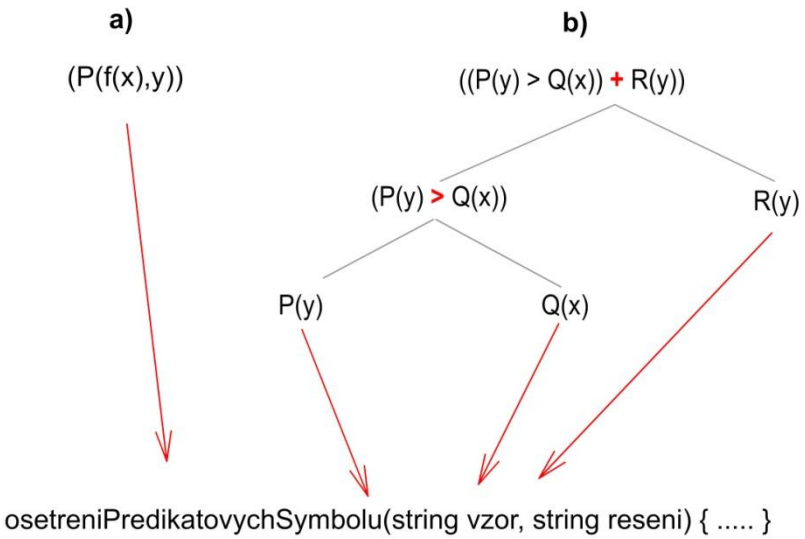
V předchozím výkladu jsem se zatím ještě nezmiňoval, co se stane v případě, že hlavní logická spojka vzorové podsekvence neodpovídá logické spojce u řešení. V takovém případě nastává pouze 6 možných případů, které mohou nastat.

1. Záměna implikace a ekvivalence
2. Záměna implikace a konjunkce
3. Záměna implikace a disjunkce
4. Záměna ekvivalence a konjunkce
5. Záměna ekvivalence a disjunkce
6. Záměna konjunkce a disjunkce

V každé možné situaci, je tato chyba zaznamenána a analýza pokračuje dále rekurzivním sestupem pro další ošetřování částí formule tak, jako to bylo popsáno výše.

Poslední dosud nepopsanou částí průběhu celého algoritmu je ověření správnosti atomických predikátových symbolů. Jak už bylo dříve naznačeno, tato funkce se volá, když nastane jedna ze situací:

a) Celá formule neobsahuje žádnou logickou spojku a funkce se volá ještě před samotným rekurzivním procházením formule pro separaci jednotlivých (neatomických) částí.
b) Funkce se volá, až když není možný další sestup při separaci částí formule.



Obrázek 23: Případy volání funkce pro ošetření atomického predikátového symbolu

Nutno podotknout, že jak už deklarace funkce na obrázku dole napovídá, očekává vždy dva parametry – vzor a řešení – tedy řetězce, které bude navzájem porovnávat.

Při vstupu do funkce se nejdříve vypočítá začátek samotného těla predikátového symbolu. Tělem jsou myšleny všechny elementy, které jsou vnořeny uvnitř závorek predikátového symbolu. To

mohou být pouze symboly pro proměnné, konstanty anebo funkce. Jakmile se spočítá pozice počátku těla predikátového symbolu, prochází se tato podsekvence tímto způsobem:

Jakmile narazí na levou závorku, pak inkrementuje proměnnou uchovávající v paměti toto číslo. Jestliže narazí na závorku pravou, toto číslo dekrementuje. Jestliže během průchodu narazí na dělící symbol pro parametry predikátového (resp. funkčního) symbolu (v našem případě je to čárka „ , “), pak ověří, zdali je proměnná pamatující si počet závorek rovna 0. Jestliže během průchodu těla predikátového symbolu narazí na dělící symbol splňující tuto podmínku, pak predikátový symbol obsahuje 2 parametry. Jestliže na dělící symbol nenarazí, obsahuje predikátový symbol pouze jeden parametr. Více než dva parametry predikátový symbol obsahovat nemůže, jelikož je celá dosavadní problematika vztahována pouze na binární (či unární) relace.

Následně se počty parametrů u predikátových symbolů porovnají, a jestliže si nejsou navzájem rovny, je to považováno za chybu, která je definována jako chybný počet parametrů u predikátového symbolu. Jelikož není možné v takové situaci dále porovnávat oba predikátové symboly, ošetření už dále neprobíhá a vrací pouze identifikátor této chyby.

Jestliže si jsou počty parametrů navzájem rovny, pak se bude jednat o jeden z následujících dvou scénářů:

1. Predikátový symbol obsahuje pouze jeden parametr.
2. Predikátový symbol obsahuje dva parametry.

V prvním případě je ošetřování samozřejmě jednodušší. Porovnají se navzájem parametry obou vstupních řetězců a zjišťuje se, o jaký symbol se jedná. Buď to bude proměnná, konstanta anebo funkce. V prvních dvou případech je ověřování triviální, ve třetím je situace poměrně komplikovanější, jelikož funkce může v sobě (podobně jako predikátový symbol) obsahovat jeden nebo dva další parametry, jejichž správnost je nutné taktéž ověřit. Chyb, kterých může nastat je tedy trochu více, než by se dalo na první pohled čekat. V lepším případě se vzor a řešení budou lišit v pojmenování proměnné nebo dojde k záměně konstanty za proměnnou. Toto je považováno za chybu, avšak ne tolik váženou, jako když dojde k záměně proměnné (konstanty) za funkční symbol (anebo obráceně). Větší vážnost je dána tím, že již následně není možné ověřovat správnost těla funkce, kde může docházet k dalším substitucím proměnných či konstant, které by vedly k dalším chybám. V případě, že je však u vzoru i řešení použit (správný) funkční symbol, zanořuje se algoritmus o úroveň níže a začíná kontrolovat samotné parametry funkčních symbolů. Ověřování funkčního symbolu má v podstatě podobný scénář, jako ověřování predikátového symbolu s tím rozdílem, že funkce již ve svém těle nebude obsahovat další funkční symboly. Opět tedy mohou nastat tyto dva scénáře:

1. Funkční symbol obsahuje pouze jeden parametr.
2. Funkční symbol obsahuje dva parametry.

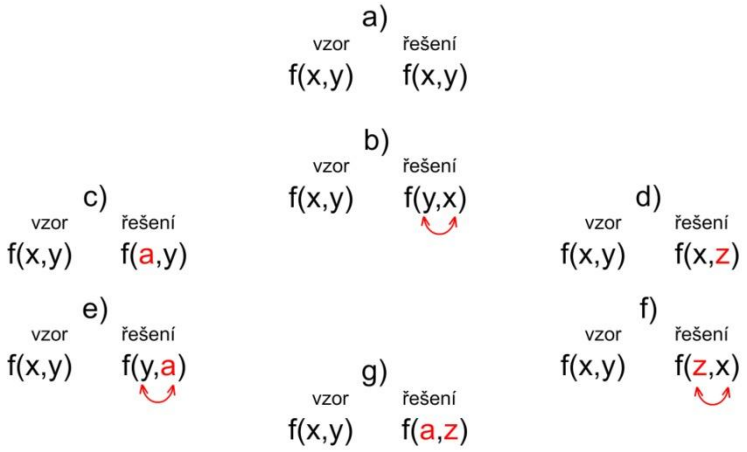
Jestliže obsahuje pouze jeden parametr, pak se tyto jednoduše navzájem porovnají a jestliže si nejsou rovny, je toto považováno za chybu, avšak ne tolik váženou (jako například chyba v pojmenování proměnné nebo záměny konstanty za proměnnou u predikátového symbolu). V případě, že funkce obsahuje 2 parametry, pak se tyto parametry navzájem ověřují a dojít může pouze k těmto očekávaným situacím:

a) Oba parametry funkce navzájem odpovídají.
b) Parametry funkce jsou navzájem přehozené.
c) Parametry funkce nesouhlasí na první pozici.
d) Parametry funkce nesouhlasí na druhé pozici.

e) Parametry funkce jsou přehozené a navíc se neshoduje první parametr u vzoru s druhým parametrem v řešení
f) Parametry funkce jsou přehozené a navíc se neshoduje druhý parametr u vzoru s prvním parametrem v řešení
g) Oba parametry jsou špatně

Jednotlivé parametry funkce (vzoru i řešení) se uloží do pole, jehož počet prvků se rovná počtu parametrů funkce. Samotné prvky pole se pak navzájem porovnávají a ošetřují všechny možné případy a) – g). V případě a) se samozřejmě nic neděje. Jestliže nastane situace popsána v bodě b), pak je toto považováno za chybu. V případě c) a d) se jedná o stejnou váhu chyby, jako byla popsána u funkce s jedním parametrem. Chyby typu e) a f) se vlastně skládají ze součtu koeficientu pro chyby z bodu b) a jednu z chyb c) a d). Jestliže jsou oba parametry špatně (bod g)), je to dvojnásobná chyba v parametrech a její váha je teoreticky stejná, jako kdyby byla celá funkce špatně. Když už je v predikátovém symbolu na počátku ověřování jeho parametrů funkce volena správně a vstoupí se do jejího těla, neměl by součet vzniklých chyb překročit váhu chyby, jako když řešitel zvolil špatný parametr u predikátového symbolu na místě, kde podle vzoru měla funkce být.

Následující obrázek ukazuje příklady jednotlivých chyb.



Obrázek 24: Druhy chyb u ošetřování funkčního symbolu

Předchozí úsek pojednával o predikátovém symbolu s jednou proměnnou a všemi možnostmi, které můžou nastat. U predikátového symbolu se dvěma parametry je situace komplikovanější. Jak už bylo psáno, ošetřování funkčního a predikátového symbolu je velmi podobné, ale s tím rozdílem, že u predikátového symbolu se mohou na místě jednotlivých parametrů objevit funkční symboly, což v případě těla funkce už možné není. Shrneme tedy všechny situace dohromady. V případě, že predikát obsahuje 2 parametry, pak se tyto parametry navzájem ověřují a dojít může pouze k těmto očekávaným situacím:

a) Oba parametry predikátu navzájem odpovídají.
b) Parametry predikátu jsou navzájem přehozené.
c) Parametry predikátu nesouhlasí na první pozici.
d) Parametry predikátu nesouhlasí na druhé pozici.
e) Parametry predikátu jsou přehozené a navíc se neshoduje první parametr u vzoru s druhým parametrem v řešení
f) Parametry predikátu jsou přehozené a navíc se neshoduje druhý parametr u vzoru s prvním parametrem v řešení
g) Oba parametry jsou špatně

Nejdříve se uloží oba parametry predikátu do pole (počet prvků je roven aritě predikátového symbolu). Samotné prvky pole se pak navzájem porovnávají a ošetřují všechny možné případy a) – g). V případě a) nenastává prozatím žádná chyba. Jednotlivé prvky v poli se analyzují, a jestliže obsahují funkční symboly, pak jsou tyto podrobeny dalšímu ověřování na správnost parametrů funkčního symbolu, jak bylo popsáno výše. V případě b) je toto považováno za chybu, avšak opět se ověřuje, o jaké symboly na místě parametrů predikátového symbolu se jedná, aby mohlo v případě nalezení funkčního symbolu dojít k dalšímu ověřování. Možnost c) a d) je trochu jiná. Jestliže se parametry predikátového symbolu na dané pozici neshodují, pak se ověřuje, k jaké záměně tam došlo. I tato situace už byla popsána dříve, ale trochu jí přiblížíme. Jelikož se jedná o chybu v případě jednoho parametru, pak se ověřuje, co za chybu to vlastně nastalo. V lepším případě se vzor a řešení liší v pojmenování proměnné anebo dojde k záměně konstanty za proměnnou. V tom horším dochází k záměně proměnné (konstanty) za funkční symbol (anebo obráceně). V druhém případě je váženost chyby větší, protože nejsme schopni hlouběji analyzovat, co se mohlo stát v případě, kdyby řešitel například správně zvolil funkci obsahující další parametry, u nichž by se mohly vyskytnout další chyby. Jakmile je tento chybný parametr u predikátového symbolu ošetřen, ověřuje se dále, co za symbol se ukrývá pod tím správně zvoleným na první (resp. druhé) pozici u predikátového symbolu. Jestliže se jedná o funkci, podrobuje se opět další analýze pro ověřování parametrů funkce. Možnosti e) a f) jsou velmi podobné možnostem c) a d), zde je akorát přičtena ke koeficientu chyb další hodnota (přehození parametrů predikátového symbolu). Kromě toho je potřeba si u ověřování parametrů predikátových symbolů dávat pozor, které prvky z tabulky obsahující jednotlivé symboly (proměnných, konstant a funkcí) se navzájem budou porovnávat. Jestliže nastane možnost g), tedy nejhorší varianta, je vrácen identifikátor pro chybný predikát.

5. Závěr

Na počátku této práce byla teoretická část zaměřena především na výklad sémantiky predikátové logiky 1. řádu. Čtenář se mohl blíže seznámit se základními principy utváření logických formulí, zákonitostmi a vztahy mezi jednotlivými elementy vstupující do dané problematiky. Blíže bylo prezentováno, co je to interpretační struktura logických formulí a dále pak představeny rozdíly mezi tím, co ve skutečnosti znamená, že je formule pravdivá (resp. nepravdivá), splnitelná (resp. nesplnitelná) v dané interpretaci. Následně byly objasněny pojmy množiny, relace a s nimi související pojmy jakožto i funkce, jejich vztah k relacím a množinám a také jejich možné druhy.

Po teoretické části se dále čtenář mohl blíže seznámit s problematikou automatizovaného přístupu při vyhodnocování logických formulí. Představena byla studie pojednávající o nejčastějším chybování studentů, která byla pořízena na základě velmi velkého datového souboru. Na jejím základě pak byla realizována konkrétní metrika vztahující se k výuce matematické logiky na této fakultě, která bere v potaz všechny přípustné chyby, kterých se studenti ať už ve větší či menší míře dopouštějí, a tedy by měl být na tyto partie brán větší zřetel ve výuce.

V poslední části je poté popsán teoreticky celý algoritmus, který na základě vzorového řešení ověřuje podobnost podaného řešení studentem. Ačkoliv algoritmus nezohledňuje De-Morganovy zákony, zákony pro implikaci a jiné další převody mezi formulemi, dokáže ověřit mnoho faktorů a do jisté míry se ekvivalentně přizpůsobit studentovu řešení a tím pádem se snažit pojmout co nejvíce jeho postup při formalizaci vět z přirozeného jazyka do jazyka matematické logiky.

Do budoucna je určitě možné počítat s možným rozšířením stávající konstrukce ověřovacího systému formulí. Mám na mysli především ty části, které v současném stavu chybí, jako je ošetření všech možných ekvivalentních zápisu formulí nebo odstranění podmínky, která vyžaduje, aby byla formule v prenexním tvaru. Kromě těchto rozšíření bych se v budoucnu více snažil zaměřit na zobecnění celého algoritmu tak, aby se nevztahoval tolik na pevně danou gramatiku a pravidla, která jsem si vymezil v rámci této práce.

6. Literatura

[1] DUŽÍ, Marie. Matematická logika [online]. Ostrava : Vysoká škola báňská, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2003 [cit. 2010-05-03]. Dostupné z WWW: <http://www.cs.vsb.cz/duzi/Matlogika.pdf>.

[2] BĚLOHLÁVEK, Radim; VYCHODIL, Vilém. Diskrétní matematika pro informatiky I. [online]. Olomouc : Univerzita Palackého, Přírodovědecká fakulta, Katedra informatiky, 2006 [cit. 2010-05-03]. Dostupné z WWW: <http://www.kubaz.cz/texty/BelohlavekDMI.pdf>.

[3] BOWEN, Jonathan; HALL, Anthony. Predicate logic [online]. London : University college, Department of computer science, 2007 [cit. 2010-05-03]. Dostupné z WWW: <http://resist.isti.cnr.it/free_slides/testing/bowen/w1_l3_pred.pdf>.

[4] RACLAVSKÝ, Jiří. Predikátová logika [online]. Brno : Masarykova univerzita, Filozofická fakulta, 1999 [cit. 2010-05-03]. Dostupné z WWW: <http://www.phil.muni.cz/fil/logika/data/text_pl.pdf>.

[5] PASEKA, Jan. Množiny a relace [online]. Brno : Vysoké učení technické, Fakulta chemická, 2004 [cit. 2010-05-03]. Dostupné z WWW: <http://www.fch.vutbr.cz/~polcerova/mat1/texty/mnozrel.pdf>.

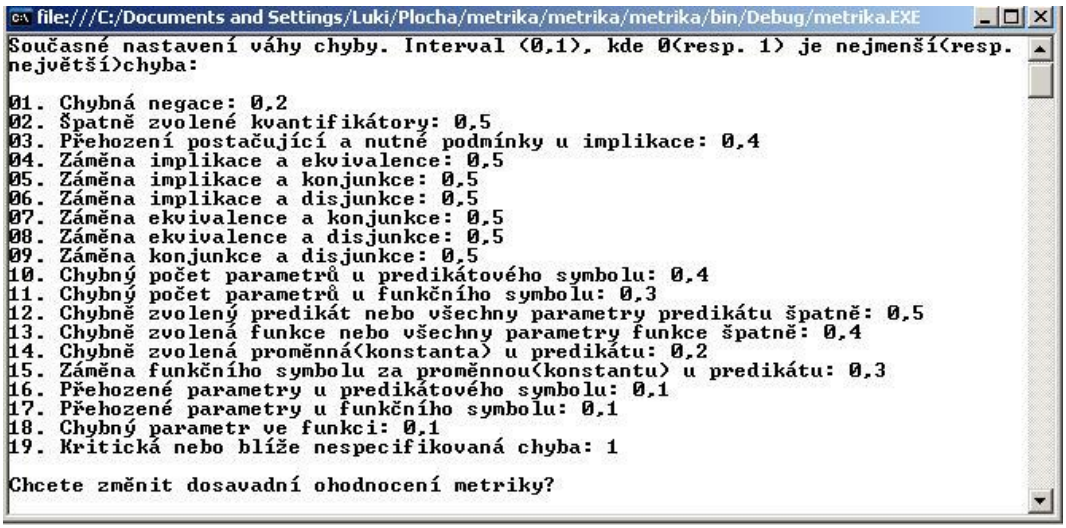
[6] HODEROVÁ, Jana. Základy logiky a teorie množin [online]. Brno : Vysoké učení technické, Fakulta strojního inženýrství, Ústav matematiky, 2006 [cit. 2010-05-03]. Dostupné z WWW: <http://www.kubaz.cz/texty/VUTMI.pdf>.

[7] BARKER-PLUMMER, Dave; COX, Richard; DALE, Robert Language, Proof and Logic.. In An empirical study of errors in translating natural language into logic. Sydney : Macquarie university, Faculty of science, 2008 [cit. 2010-05-03]. Dostupné z WWW: <http://web.science.mq.edu.au/~rdale/publications/papers/2008/fp273Barker-Plummer.pdf>.

[8] Menšík, M., Číhalová, M., Jarotek, V.: Interpretace logických formulí v PL1 do přirozeného jazyka. In Odkud a jak brát stále nové příklady? Elektronická databáze příkladů pro výuku logiky na VŠ.. Ed. Dostálová & kol., Plzeň:Západočeská univerzita, 2009, p. 85-94, ISBN 978-80-7043-864-0

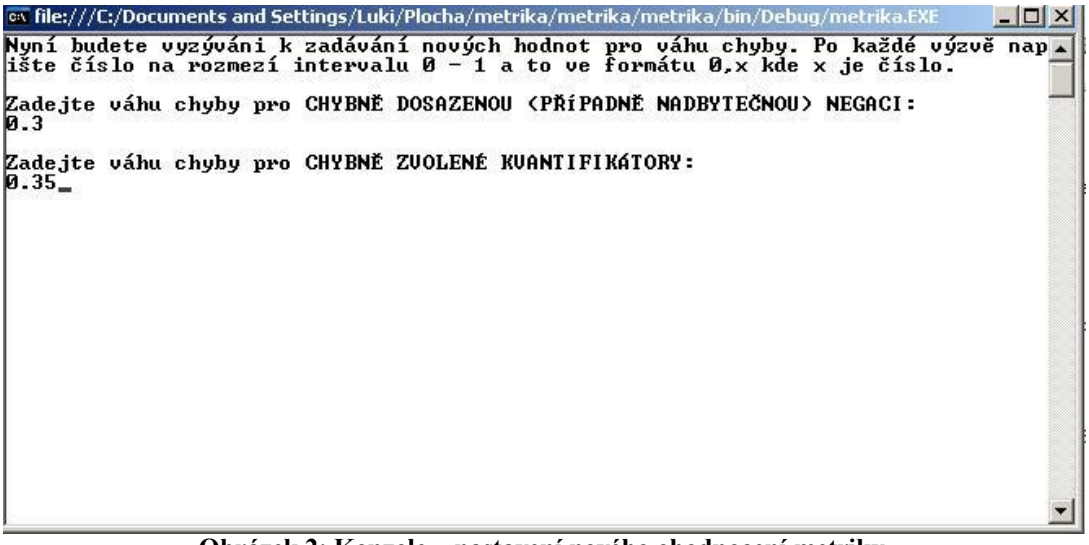
Příloha I. – Uživatelská příručka k programu

Program Metrika se spouští dvojitým kliknutím na spustitelný soubor metrika.exe (lokální cesta k souboru: Složka programu/metrika/bin/debug/metrika.exe). Po najetí aplikace v konzoli nejdříve program očekává, že mu uživatel sdělí, jestli chce změnit parametry pro váhu jednotlivých očekávatelných chyb (viz. níže). Uživatel může zadat buď souhlas (do řádku napíše „a“ nebo „ano“ a zmáčkne enter) nebo nesouhlas v případě, že chce zachovat dosavadní ohodnocení chyb, které je znázorněno na konzoli (do řádku nenapíše nic, případně napíše cokoliv jiného než „a“ nebo „ano“ a zmáčkne enter).



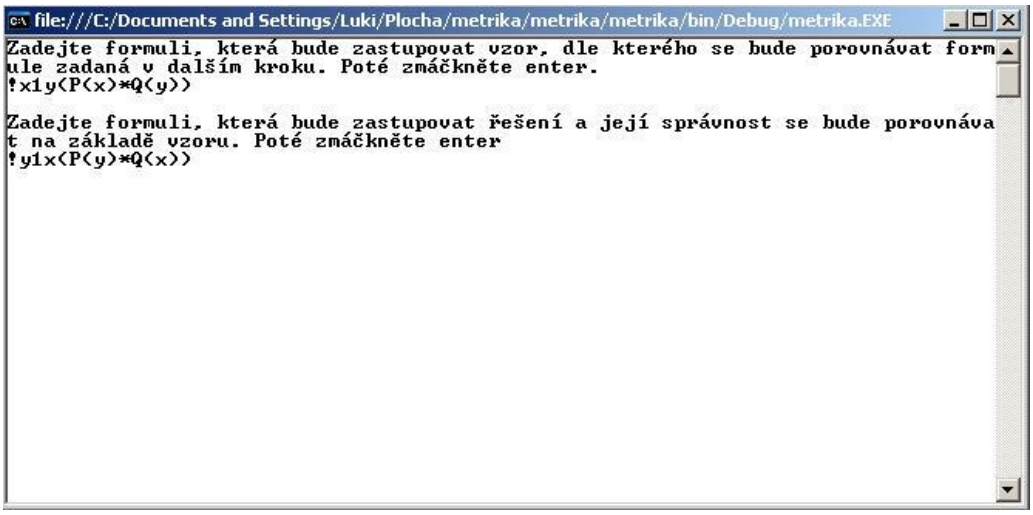
Obrázek 1: Konzole - aktuální ohodnocení metriky

V případě první volby (že zadal uživatel souhlas pro změnu nastavení), se na konzoli vypíšou další pokyny pro nové nastavení hodnot. Každá chyba se nastavuje samostatně a to tak, že po výzvě programu k zadání váhy chyby pro konkrétní případ čeká program vstup uživatele ve formátu čísla s desetinnou čárkou, přičemž formát zápisu čísla je 0,x, kde x je přirozené číslo (chyba nabývá své reálné hodnoty na intervalu (0,1)). Po zadání čísla zmáčkne enter a program vypíše pokyny pro ohodnocení další chyby. Takto uživatel pokračuje, dokud neohodnotí všechny uvedené případy očekávaných chyb (ukázka na obrázku níže).



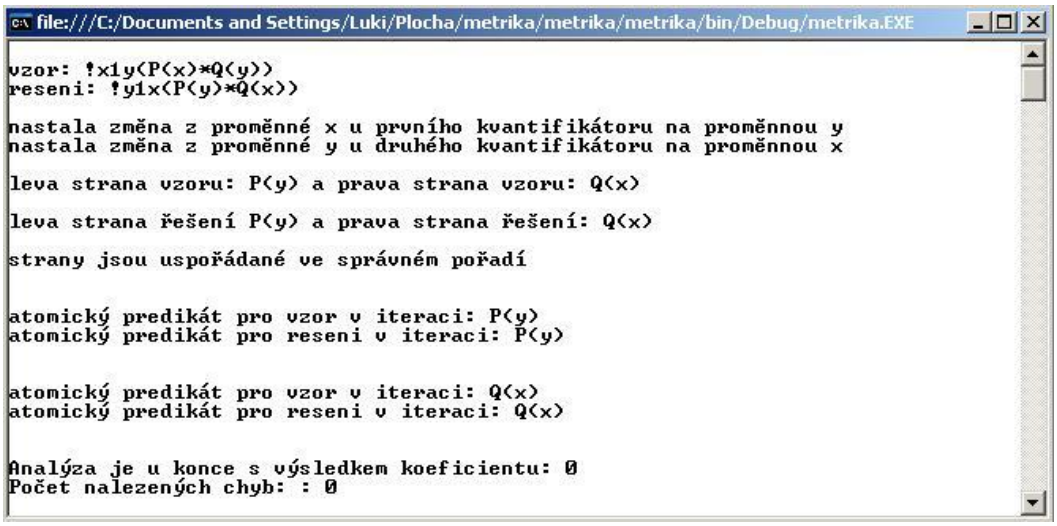
Obrázek 2: Konzole – nastavení nového ohodnocení metriky

Veškerá nově uložená data se ukládají do externího souboru (lokální cesta k souboru: Složka programu/metrika/bin/debug/ohodnoceni_metriky.txt), odkud jsou při příštím spuštění programu opět načtena. Po dokončení zápisu nových hodnot program vyzve uživatele, aby nejdříve vložil formuli, která bude prezentovat vzor. Po jejím zapsání a zmáčknutí enter bude vyzván, aby vložil druhou formuli, tentokráte samotné řešení. Po jejím zadání očekává opět enter a následuje samotný průběh programu.

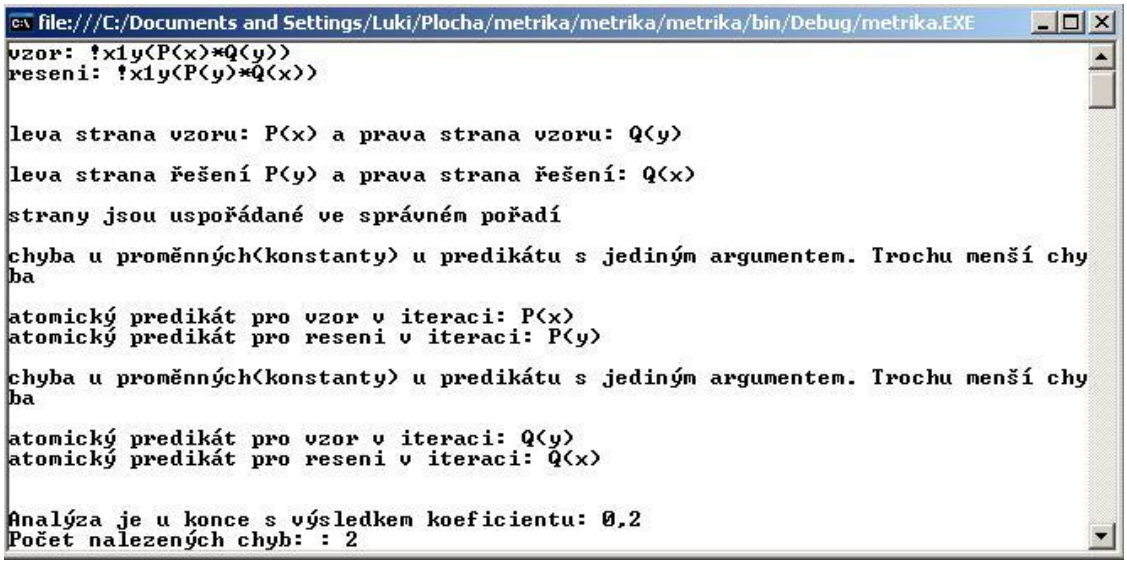


Obrázek 3: Konzole – vložení formulí

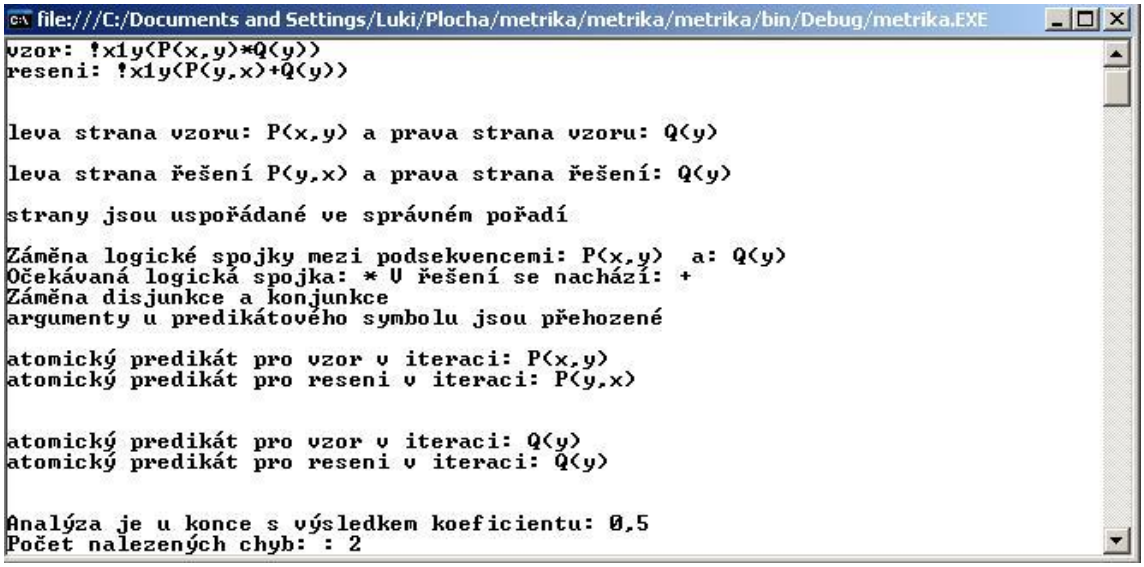
Program následně rozparsuje jednotlivé části obou formulí a podrobí je analýze. Na konci vypíše počet nalezených chyb a kromě nich i koeficient představující součet všech ohodnocení nalezených chyb.



Obrázek 4: Konzole – proces ošetření formule



Obrázek 5: Konzole – proces ošetření formule



Obrázek 6: Konzole – proces ošetření formule